Software Security Metrics for Malware Resilience

Dissertation
zur
Erlangung des Doktorgrades (Dr. rer. nat.)
der
Mathematisch-Naturwissenschaftlichen Fakultät
der
Rheinischen Friedrich-Wilhelms-Universität Bonn

vorgelegt von
Hanno Langweg
aus
Bonn

Bonn 2007

Angefertigt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Rheinischen Friedrich-Wilhelms-Universität Bonn

Diese Dissertation ist auf dem Hochschulschriftenserver der ULB Bonn http://hss.ulb.uni-bonn.de/diss_online elektronisch publiziert.

Erscheinungsjahr: 2008

1. Referent: Prof. Dr. Armin B. Cremers
2. Referent: Prof. Dr. Einar Snekkenes

Tag der Promotion: 25.02.2008

Software Security Metrics for Malware Resilience

by

Hanno Langweg

Submitted to the Mathematisch-Naturwissenschaftliche Fakultät
on 2007-09-26, in partial fulfillment of the
requirements for the degree of
Dr. rer. nat.

Abstract

We examine the level of resistance offered by a software product against malicious software (malware) attacks. Analysis is performed on the software architecture. This is available as a result of the software design process and can hence be used at an early stage in development.

A model of a generic computer system is developed, based on the internationally recognized Common Criteria for Information Technology Security Evaluation. It is formally specified in the *Z* modeling language. Malicious software attacks and security mechanisms are captured by the model. A repository of generic attack methods is given and the concept of resistance classes introduced to distinguish different levels of protection. We assess how certain architectural properties and changes in system architecture affect the possible resistance classes of a product.

This thesis has four main contributions: A generic model of an operating system from a security perspective, a repository of typical attack methods, a set of resistance classes, and an identification of software architecture metrics pertaining to ordered security levels.

Thesis Supervisor: Prof. Dr. Armin B. Cremers
Institut für Informatik III, Rheinische Friedrich-Wilhelms-Universität Bonn

Thesis Supervisor: Prof. Dr. Einar A. Snekkenes
Avdeling for informatikk og medieteknikk, Høgskolen i Gjøvik

Contents

1 Introduction **13**
1.1 Software evaluation . . . 13
1.2 Architecture . . . 14
1.3 Scope . . . 14
1.4 Research questions . . . 15
1.5 Contributions . . . 15
1.6 Overview . . . 16

2 Previous and related work **17**
2.1 Software and system architecture . . . 17
2.2 Software metrics . . . 22
2.3 Security metrics . . . 27
2.4 Metrics in security evaluation criteria . . . 34
2.5 System specification . . . 41

3 Software metrics for resilience **45**
3.1 Ranking of security requirements . . . 46
3.2 Attacker capability metrics . . . 48
3.3 Generic attacks . . . 50
3.4 Resistance classes . . . 52
3.5 Properties of secure software architectures . . . 53
3.6 Software architecture metrics . . . 57

4 Model of a generic computer system **71**
4.1 Modeling approach . . . 71
4.2 Scope and elements of model . . . 74
4.3 Architectural description . . . 82
4.4 Example of an architectural description . . . 84

5 Formal model: Computer system **87**
5.1 Formal specification in Z . . . 87
5.2 Model definition . . . 87
5.3 Limitations of the model . . . 125
Model definitions index . . . 127

6 Formal model: Attacks **133**
6.1 Repository of generic malware attacks . . . 133
6.2 Security requirements . . . 165

6.3 Attacker capabilities 166
6.4 Resistance classes 168

7 Architectural analysis 179
7.1 Architectural changes 179
7.2 Homebanking with FinTS/HBCI 179
7.3 Discussion 189

8 Conclusions 193
8.1 Contributions 193
8.2 Discussion 195
8.3 Future work 196

A Security Checklists 197

B StarMoney architecture 209

References 239

Index 249

Curriculum vitae 251

List of Figures

4-1 Computer system model based on selected Common Criteria security functional requirements . . . 73
4-2 Executable data components and execute connectors of `SCRSetup.exe` . . . 85
4-3 Executable data components of `SCRSetup.exe` interfacing with the local human user . . . 86

7-1 Dependency graph for executable modules used by `StarMoney.exe`, part (i) 182
7-2 Dependency graph for executable modules used by `StarMoney.exe`, part (ii) 183
7-3 Dependency graph for executable modules used by `StarMoney.exe`, part (iii) 184
7-4 Dependency graph for executable modules used by `StarMoney.exe`, part (iv) 185
7-5 Executable data components and execute connectors of `StartStarMoney.exe` 186

List of Tables

2.1 Metric Detail Form according to NIST SP 800-55 [NIS03] 28
2.2 SQM security criteria definitions 31
2.3 CCI weights for evaluation questions 31
2.4 Classes in the TCSEC 35
2.5 Calculation of attack potential (based on table [CEM04] B.8.3) 39
2.6 Rating of vulnerabilities (based on table [CEM04] B.8.4) 40
2.7 EALs and attack potential (based on tables [CC299c] B.19 and [CEM04] B.8.1, B.8.2 40
2.8 EALs and attack potential (based on tables [CC305c] E.24 and [CEM05] B.3 41

3.1 Categories for generic attacks (cf. [CEM04]) 51
3.2 Generic malware attack methods 52
3.3 Metric M_1: Restriction of number of executable distribution sources 60
3.4 Metric M_2: Restriction of number of executable components 61
3.5 Metric M_3: Percentage of protected executables 62
3.6 Metric M_4: Percentage of protected intermediate storage components . . . 62
3.7 Metric M_5: Percentage of access control instrumentation 63
3.8 Metric M_6: Conformity of access permissions 63
3.9 Metric M_7: Percentage of logged invocations 64
3.10 Metric M_8: Percentage of authenticity/integrity preserving connectors . . . 64
3.11 Metric M_9: Percentage of unlogged security parameters 65
3.12 Metric M_{10}: Restriction of number of components with shared responsibility (server) 65
3.13 Metric M_{11}: Restriction of number of components with multiple executable extensions 66
3.14 Metric M_{12}: Percentage of trusted path connectors 67
3.15 Metric M_{13}: Restriction of number of privileges 68
3.16 Metric M_{14}: Restriction of number of processes sharing a privilege 68
3.17 Software metrics for secure architectures 69
3.18 Correspondence between architectural properties and security metrics . . . 70

4.1 Common Criteria families used to build model 74
4.2 Unused Common Criteria classes to build model 75

6.1 Necessary capabilities for attacks in repository 167

7.1 Values for security metrics applied to StarMoney 5.0 191
7.2 Comparison of metrics values for StarMoney 5.0 and 6.0 192
7.3 Tool support for Z 192

Chapter 1

Introduction

When you can measure what you are speaking about,
and can express it in numbers, you know something about it;
but when you cannot measure it, when you cannot express it in numbers,
then your knowledge is of a meagre and unsatisfactory kind.
— Lord Kelvin, quoted in [She95]

This thesis has four main contributions: A generic model of an operating system from a security perspective, a repository of typical attack methods, a set of resistance classes, and an identification of software architecture metrics pertaining to ordered security levels.

1.1 Software evaluation

Early evaluation of the security of software-intensive systems was based on penetration efforts by skilled individuals or groups. There was usually little structure and documentation involved in the process, and tests proved the presence of defects rather than their absence. Checklists later helped manufacturers and operators to build and configure systems with respect to typical security requirements.

Security evaluation criteria were developed with the aim of a structured and repeatable process for determining whether a software system lived up to its security requirements. Early approaches introduced a hierarchy of requirements, while later attempts focused more on scales for how rigorous the evaluation process was. Comprehensive evaluation, especially when done rigorously, is still expensive and requires a lot of time. Repositories of standardised software attacks barely exist – opposed to other disciplines, e.g., fire protection or physical security. Much relies on an evaluator's experience. Often, product evaluations are completed when the next version of the product is announced or is being shipped already. Requirements are often not comparable among evaluations, so an acquirer faced with two evaluation reports only knows that both products are evaluated, but not which product can be supposed to be more secure than the other.

State of the industry is to proclaim that "100% security" will never be available. While this is provably true as regards detection of computer viruses (cf. [Coh85]), it hides which scale people have in mind. Security requirements are not a moving target, but can be ordered. In addition, capabilities of adversaries can be ordered. It is important to determine protection at the required level of protection against the accepted level of adversary, in light of the possible attack methods. Under these constraints, a security level of "100 %" might indeed be achievable.

A full rigorous analysis of a software system's protection is often not feasible. Sometimes, source code or detailed documentation is not available. Sometimes, the system does not yet exist. Sometimes, one is interested in a class of similar systems. In these situations an analysis of a system's architecture provides an acceptable level of detail to capture fundamental defects in the design.

1.2 Architecture

The architecture of a software application is a result of the application's design process. There is no uniform definition of the term 'architecture'. In the Random House Dictionary it is defined as "the structure of anything" (cited in [Neu00]). The Oxford English Dictionary [Sim05] offers "6. Computing. The conceptual structure and overall logical organization of a computer or computer-based system from the point of view of its use or design; a particular realization of this." IEEE standard 1471-2000 [IEE00] on *Recommended Practice for Architectural Description of Software-Intensive Systems* speaks of "[t]he fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution."

For our purposes, we define architecture as a system's implementation-independent structure given by its components and the relations among them and to the environment.

Architectural weaknesses are then weaknesses in the design of a system (e.g., missing/weak authentication, bypassing of controls, poor choice of parameters). They are a mismatch between security requirements and system specification and exist regardless of the quality of an architecture's implementation. In contrast there happen to be errors in an implementation (e.g., buffer overflows, validation errors) that are popular contemporary attack methods.

Attacks on a system's implementation-independent architecture may be harder to detect and harder to correct by simple means after deployment.

1.3 Scope

Our focus is on local malicious software attacks, e.g., Trojan horse programs running on a workstation or desktop computer. We are concerned with security-sensitive programs used by a local human operator.

Processes might be executed with the same privileges based on a user account. Still, an attacker can gain advantages from attacking another process. Differences are present in data that is only available to one process, e.g., from user input, communication with a server, or a hardware token. Connections could be limited between processes and the user, a server, or a hardware token. A desktop firewall software could assign different privileges to a process. The general concept here is that of a *protected subsystem* (cf. [SS75]).

An adversary does not have direct physical access and attacks are carried out by other processes on the device. We are not concerned with distributed or mobile applications; attacks on distributed systems or via network connections are outside the scope of this thesis. Local malicious software attacks may take place at the endpoints (nodes) of a network, though.

1.4 Research questions

The question addressed by this dissertation is "Can resilience against malware attacks be assessed and rated, and if so, how?"

This leads to six sub-questions:

1. What are possible axes and scales to define a security level?
2. How can a software's architecture be described, together with the surrounding operating environment and an adversary?
3. Which properties of an architecture can be measured and related to resilience against attacks?
4. How can attacks be enumerated against which protection is required?
5. What formal basis can be used to express and derive a security level?
6. Which metrics are significant and in which order should they be applied to evaluate a product?

1.5 Contributions

Resilience against malware attacks can be assessed and rated at the architectural level of a software product.

An architectural description of a product can be analysed whether it complies with the desired security requirements in presence of a capable adversary. Metrics for certain architectural properties can be used to determine if a product could achieve a higher security level by applying some changes to its architecture.

We can claim that security requirements and attacker capabilities for local malware attacks can be *formally described*. Our extended *model of a computer system* is based on the Common Criteria for Information Technology Security Evaluation (CC). With it we are able to describe and analyse attacks of local malicious processes on fundamental software design vulnerabilities. A repository of generic attacks helps to check whether a given system complies with a resistance class. Our model is flexible enough to incorporate future attack methods that are not known today.

We are able to prove that a *repository of generic attack methods* allows enumeration of possible attacks. Our derivation of this set of possible attacks on software architectural vulnerabilities is based on a complete enumeration of attack vectors for a Turing machine. It is cross-checked with unordered attack catalogues found in evaluation manuals.

Resistance classes are based on a hierarchy of security requirements and a hierarchy of attacker capabilities. Both hierarchies can be expressed by a lattice. The order on resistance classes allows to compare similar systems.

In total, 14 *software security metrics* are identified that represent the degree of compliance of a software architecture with established security principles. The metrics are associated with resistance classes so that they indicate where changes could be applied to the product's architecture to reach a higher resistance class.

1.6 Overview

We discuss relevant previous work in the next chapter. Chapter 3 defines *resistance classes* and software *metrics* for resilience of applications against attacks. Our *model of a computer system*, based on the *Common Criteria* for Information Technology Security Evaluation [CC299b], is developed in chapter 4 and introduced in the formal language *Z* in chapter 5. Attacks are formally specified in chapter 6. Metrics are validated and exemplified in terms of the model in chapter 7. Chapter 8 summarises our conclusions. The appendices contain checklists covering important mechanisms applicable to protect against malware attacks and they contain raw data about the architecture of a sample product.

Chapter 2

Previous and related work

Chapter summary: In this chapter we review the relevant previous work on measuring malware resilience of software. Software security metrics in particular have not been researched thoroughly in the past. There have been approaches to measuring properties of software artifacts in a number of related fields: software and system architecture, software metrics, security metrics, security evaluation criteria, specification techniques. Principles of how to design reliable and secure software are based on several decades of experience with software-intensive systems. Software metrics often focus on properties like software size and complexity. Frameworks for security metrics give a structure for metrics work, but do not offer guidance in how to find relevant metrics. Many security metrics operate on a very abstract level (e.g., "percentage of machines that are hardened against attacks") or on a very detailed level (e.g., length of cryptographic keys). Evaluation criteria offer a good point to start development of new security metrics. Two drawbacks are that they typically use a rough classification of adversaries (*low*, *medium*, *high* scale), and often only the level of assurance is reported, i.e., the quality of the evaluation, not the quality of the security mechanisms. There are plenty of specification techniques for formally modeling systems and software. They introduce clarity in the presentation, remove ambiguities, and support mathematical proofs of software properties.

In the next sections we review relevant work related to software architecture, software and security metrics, and formal system specification. Literature is selected with a focus on malicious software and non-distributed computer systems, e.g., workstations or personal computers.

2.1 Software and system architecture

2.1.1 Saltzer and Schroeder (1973/1975). The Protection of Information in Computer Systems

This paper is one of the early fundamental efforts in computer security. [SS75] It was first presented at the 1973 Symposium on Operating System Principles and later revised. It gives an indication of the state of the art at its time. In its first part, design principles for secure systems are given that still hold for the development of secure systems today and have found their way into textbooks. No essentially different design principles have arisen after this article. The value lies not in each principle itself – most had previously

been published – but in their collection and application to protection.
The eight principles that apply particularly well to protection mechanisms are

1. *Economy of mechanism* – keep the design as small and simple as possible to facilitate evaluation and maintenance.
2. *Fail-safe defaults* – base access decisions on permission rather than exclusion, so that there is no lack of protection in case of error.
3. *Complete mediation* – every access to every object must be checked for authority, so that circumvention of access control is not possible.
4. *Open design* – the design should not have to be secret to provide security; however, it can additionally be kept secret.
5. *Separation of privilege* – two or more conditions must be met before access should be permitted, so that two or more subjects assume responsibility.
6. *Least privilege* – every subject should operate using the least set of privileges necessary to complete the task to minimize accidental or deliberate improper use of privileges.
7. *Least common mechanism* – minimize the amount of mechanism common to more than one subject and depended on by all subjects, so that no information is shared along this mechanism without authorization, and to ease certification of mechanisms.
8. *Psychological acceptability* – the human interface must be easy to use, so that users routinely and automatically apply the protection mechanisms correctly.

Two more design principles are named that the authors consider to apply only imperfectly to computer systems:

- *Work factor* – the cost of circumventing a protection mechanism, e.g., trying all possible character combinations of a password. Many computer protection mechanisms are not susceptible to direct work factor calculation, since defeating them by systematic attack may be logically impossible.
- *Compromise recording* – mechanisms that reliably record that a compromise of information has occurred instead of more elaborate mechanisms that completely prevent loss. This is used rarely, since it is difficult to guarantee discovery once security is broken.

The authors stress that their design principles do not present absolute rules. Their violation should indicate potential problems and trigger a careful review of the system, an approach taken by [Hog88].

2.1.2 Gasser (1988). Building a Secure Computer System

This book on building secure computer systems [Gas88] devotes a whole chapter to principles of a security architecture. These principles are similar to a selection of the the classical stated by [SS75]: *Minimize and isolate security controls* (i.e., Least common mechanism), *Enforce least privilege* (i.e., Least privilege), *Structure security-relevant functions* (cf. Economy of mechanism), *Make security friendly* (i.e., Psychological acceptability), *Do not depend on secrecy for security* (i.e., Open design).

2.1.3 Hogan (1988). Protection Imperfect: The Security of Some Computing Environments

This paper [Hog88] examines the Unix operating system if it fulfils Saltzer's and Schroeders's principles [SS75] for the design of secure systems. Of the eight principles, three are assigned lower priority in the study. Separation of privilege is supposed not to exist in systems owing to its difficulties in implementation. Even insecure systems are claimed to comply with Open design and Fail-safe defaults.

As a result, numerous weaknesses in the design of Unix are exposed. Achieving a secure system is said to be hard for users and administrators as it is hard to mitigate the vulnerabilities shown. Designers and implementors are seen to be in the most effective position to deal with security problems.

2.1.4 Schneider (1999). Security Architecture-Based System Design

The architecture described in this paper [Sch99] is not really an architecture. It is more like a security model that is very similar to a modified discretionary access control model. [SCL00] The paper distinguishes system, software, and information architecture. The information architecture introduces information domain, principals and structures for information transfer between domains that can be regulated by a security policy.

2.1.5 Neumann (2000). Report on Practical Architectures for Survivable Systems and Networks

This report [Neu00] discusses architectural properties and requirements for survivable systems. The notion of survivability encompasses the traditional topics of computer security, i.e., confidentiality, integrity, and availability. It is defined as the ability of an application to satisfy and continue to satisfy critical requirements in the face of adverse conditions.

The important section *Architectures for survivability* lists structural organizing principles. These are similar to those stated in the classical paper of Saltzer and Schroeder [SS75]. They are quite general and provide guidance of use in the design and implementation phase of a system. Their generality makes it difficult to apply them directly in an analysis of whether they have been applied and how they influence security of a system in detail.

The topic is then further elaborated with a consideration of *architectural structures* and *architectural components*. The former, architectural structures, can be regarded as concepts to structure systems and are useful at various levels of detail (the report focuses on large systems rather than smaller application programs, though). The latter, architectural components, are building blocks to build large and complex structures. Concepts to securely structure computer systems comprise hierarchical structures, protection domains, security kernels and trusted computing bases, separation kernels, isolation mechanisms, read-only bootstraps and backups, multilevel-security and multilevel-integrity trusted computing bases, selective-trustworthiness architectures, thin-client end-user architectures, explicitly compensating systems structures, minimum essential information infrastructures, and concepts from classical control theory.

Building blocks to achieve secure structures encompass secure operating systems, encryption and key management, authentication, trusted paths and resource integrity, servers/services, wrappers, protocols, firewalls and routers, and monitoring. These components might be used to partition a software under review in its elements.

It is noted in section 7.3.12 of [Neu00] that survivability and its subtended requirements of security and reliability are fundamentally weak-link problems. Without further justification a collection of attack methods is provided by table 7.1: lack of correctness, no reliability of lower layer, spoofing/simulation, flawed protocol design, flawed code, poorly embedded cryptography, compromise from below/within/outside, reliance on fixed reusable passwords, nonexistent or weak trusted paths, bypassing, alteration of components.

In total, the report is a broad comprehensive survey of the state of the art as regards survivability of large systems. With respect to smaller application software it might be possible to re-use attack methods and some of the architectural components. The architectural structures, however, seem harder to fit with smaller problem sets.

2.1.6 Bundesnetzagentur (2001/2005). Unified specification of operating conditions for signature creation applications

The regulatory body for electronic signatures in Germany – Bundesnetzagentur, formerly Regulierungsbehörde für Post und Telekommunikation – issued a paper [Bun05] on operating conditions for signature creation applications. It is based on a workshop convening authorities, corporations, consumer watchdog organizations, and called in after doubts about the current level of security of applications for electronic signatures (cf. [SCL01a] and [SCL02]). The paper is intended as guidance for developers and evaluators of software. According to [KF05], it is the only structured attempt known so far to establish a hierarchical ordering of the hostility of an operating environment.

In the document, security requirements for signature creation applications are stated, derived from their legal basis, and a short list of possible threats is given: attacks via communication networks, attacks by manual access of unauthorized persons and data transmission by removable media, errors and manipulations under installation, operation and maintenance. Security measures are to be applied either by the signature component or the operating environment.
Three classes of operating environments are distinguished:

- *Unprotected working space* (Ungeschützter Arbeitsbereich) Unprotected connection to the Internet and no special security measures in the operating environment. This is the most challenging environment defined, and is seen as a special case that only few developers will pursue. It is likely that closed hardware solutions will fall into this category.

- *Protected working space* (Geschützter Einsatzbereich) Signature software is used at a signature workstation. Potential attacks via Internet, Intranet, manual access by unauthorized persons, data transmission by removable media are precluded by a combination of security measures in the application and the environment. This is the envisioned standard case.

- *Isolated working space* (Isolierter Einsatzbereich) Signature software is used at a signature workstation. At no time there exists a connection to a communications

network and the operating environment ensures that there will be no manual access by unauthorized persons and no data transmission by removable media. It could e.g. be achieved by the use of guards limiting access to authorized persons. This is the strongest protection offered by the environment and requires fewest security measures in the application software. It is seen as a special case.

In the standard case – protected working space –, attacks via the Internet are to be blocked, attacks via an Intranet and by manual access and removable media are to be thwarted by the operating environment, the IT platform and the signature creation application. Installation, operation and maintenance are assumed to be performed securely by qualified and trustworthy personnel and administrative measures. The latter holds for all the three environment classes. The three working space levels could be regarded as a scale for adversary capabilities.

Security measures are to be applied by the operating environment and the signature application in the standard case. They are not specified further and how responsibility is expected to be shared is not defined in the document. This is an obligation to fulfil when submitting an application for evaluation; then a detailed product-related specification is necessary. The only restriction is that high security must not be accomplished by severely limiting the permitted operating environment. A significant part has to be achieved by constructive technical methods in the software. However, it is also granted in a footnote that IT platform and applications have to be trustworthy, i.e., they especially have to be free of malicious software. It might be questioned which significant threats remain apart from attacks via an Intranet and guessing of PINs.

Since 2002, roughly a dozen certificates of conformity have been issued that relate to this document. However, all three evaluating bodies involved use a very similar language in defining requirements put on the operating environment. Access from the Internet and Intranet has to be blocked by appropriate methods and it has to be confirmed that the IT platform is free of malicious software and cannot be manipulated by unauthorized persons. Compared to certificates issued before 2002, these conditions are equivalent to earlier operating limits that were deemed unsatisfactory by the regulating authority.

While the approach made is respectable, it does not appear to be honoured by developers and evaluating bodies. [Lan06b]

2.1.7 Sample architectures from a security perspective

In [LG99] some well-known architectural styles are presented: layering, pipe-and-filter, hub-and-spoke, client/server.

A collection of 25 *architectural styles* is listed in [SC96]. They are classified according to their constituent parts, control and data issues, and the interaction between control and data flows. The styles relate more to larger system architectures and are not well applicable to single systems and malware attacks.

Problematic architecture interactions are reviewed in [DGP+01]. Interoperability problems are identified in three categories: control transfer, data transfer, and interaction initialisation. In a three-step methodology characteristics of all components are determined and then problematic interactions marked in a bipartite conflict graph. Interactions can be merged to simplify description. Analysis has to proceed manually, so the contribution mainly is in taxonomy and notation.

Literature on *security engineering* is surveyed by [SCH04]. The authors then analyse the *qmail mail server* software and how its architecture supports the security requirements for a mail server. The main principles employed are the use of least privilege, compartmentalization, economy of mechanism, and no trust placed in untrustworthy input.

In a similar approach [HJA04] identifies some known *design patterns* (Compartmentalization, Distributed Responsibility, Unique Entry of Information, Recoverable Component, Checkpointed System, Hot Standby) and derives further patterns based on qmail's architecture (Small Processes, Content-Independent Processing) and implementation (Safe Data Structure).

Middleware architectures – CORBA, J2EE, .NET – are discussed with respect to their security architecture in [GAG05]. The article focuses on specific mechanisms that fulfil security requirements assumed for distributed systems. It could be argued whether these are architectural questions or rather an overview of middleware security features.

2.2 Software metrics

Measurement is crucial to the progress of all sciences. Collection of data leads to observations and generalizations that allow derivation of theories and their confirmation or refutation via hypothesis testing. In the Oxford English Dictionary [Sim05], *metrics* are defined as

> 4. A system or standard of measurement; a criterion or set of criteria stated in quantifiable terms.

Quality of measurements is characterised by reliability and validity. *Reliability* is the consistency of measurements. One needs a good operational definition for this. *Validity* is the extent to which an empirical measure reflects the real meaning of the concept under consideration. Often, it is difficult to recognize that a certain metric is invalid in measuring a concept.

Measurements can be hierarchically grouped into different scales: nominal, ordinal, interval, ratio [Kan02]. Nominal scales allow to differentiate objects. Ordinal scales define an order on objects. Interval scales allow to determine an difference between objects. Ratio scales are interval scales that have an absolute or nonarbitrary zero point.

In [MP93] three classes of software metrics are distinguished: *size*, *product quality*, and *process quality*. In addition, there exist more proposals for structural code metrics, e.g. information flow metrics (cf. [HK81]).

2.2.1 Size and structure

Code size/complexity measures include *LOC*, McCabe's *cyclomatic complexity* and Halstead's *software science*. Other approaches are e.g. based on *function points*. These measures are easy to obtain, and can often be calculated by automated means.

LOC stands for *lines of code*. There are several different definitions for counting LOC, among them *number of source code lines*, *instruction statements*, *instruction delimiters*, *non-commentary source LOC*. Variations can range up to 500% for the same source code [She95]. This measure depends on the programming language used.

McCabe's *cyclomatic complexity* [McC76] is the number of regions in a programs control flow graph. It is computed as $e - n + 2p$ where e is the number of edges, n

the number of nodes, and p the number of disconnected components of the graph. It is also equal to the number of binary decisions in a program plus 1. For good testability and maintainability it is recommended that a program module not exceeds a cyclomatic complexity of 10. If a significant correlation between complexity and defect level can be established in an organisation, cyclomatic complexity can be used to identify complex parts of a program warranting detailed inspections and to estimate programming and service effort, as well as identify troublesome code.

Basis of Halstead's *software science* is that any programming task consists of selecting and arranging a finite number of program tokens that are basic syntactic units distinguishable by a compiler. All tokens in a computer program are regarded as either operators or operands. Halstead's measures are: n_1 number of distinct operators in a program, n_2 number of distinct operands in a program, N_1 number of operator occurrences, N_2 number of operand occurrences. Based on these, some equations for different program attributes are derived, including the total vocabulary, the overall program length, the potential minimum volume for an algorithm, the actual volume (number of bits required to specify a program), the program level (a measure of software complexity), program difficulty and other features such as development effort and the projected number of faults in the software. The work had a significant impact on software measurement and was instrumental in making metrics studies an issue among computer scientists. Still, software science has been controversial since its introduction. It has been criticized for its methodology, derivations of equations, human memory models and others. There has been little empirical support for the equations.

The idea of *function points* is to focus on the size of the requirements specification in contrast to size of source code or executable. Hence, implementation language independence is achieved. Five basic classes of functions are used: external input types (e.g. file names), external output types (e.g. reports, messages), enquiries (interactive inputs needing a response), external files (i.e. files shared with other software systems), internal files (i.e. invisible outside the system). Each function appearing in the specification is weighted by a factor according to its complexity (determined by counting files, record types and data elements involved). This yields the unadjusted function count. It is multiplied by a factor based on fourteen general system characteristics. In spite of detailed counting rules as given by e.g. [Int04], variation among analysts is likely to range around 10–30% according to [She95]. In addition, function points are a synthetic measure that is difficult to validate and interpret. Their main application is to predict software project effort.

Software project *cost estimation* aims at estimating effort and schedule to develop a software product. These measures are often normalized by using size measures (such as *KLOC*, thousands of lines of code) for implementation and other factors. A representative for this class is Boehm's *COCOMO*. [Boe81]

The *COnstructive COst MOdel* (COCOMO) is a model for software product cost estimation with three levels. The first level, *Basic COCOMO*, is a static, single-valued model that computes softwar edevelopment effort as a function of program size expressed in estimated lines of code/number of delivered source instructions (DSI). The second level, *Intermediate COCOMO*, computes software development effort as a function of program size and a set of *cost drivers* that include subjective assessments of product, hardware, personnel, and project attributes. The third level, *Advanced COCOMO*, incorporates all characteristics of the intermediate version with an assessment of the cost drivers's impact on each step of the software engineering process.

As an example, *Basic COCOMO* estimates the development effort in person months as $Effort = A * KDSI^B$ and $Development\ time = C * Effort^D$. The parameters A, B, C, D depend on the type of project, i.e., *organic*, *semi-detached*, or *embedded*.

- *Organic* projects are relatively small, simple software projects in which small teams with good application experience work to a set of less than rigid requirements. $A = 2.4$, $B = 1.05$, $C = 2.5$, $D = 0.38$.
- *Semi-detached* projects are in size and complexity intermediate software projkects in which teams with mixed experience levels must meet a mix of rigid and less than rigid requirements. $A = 3.0$, $B = 1.12$, $C = 2.5$, $D = 0.35$.
- *Embedded projects* are softwar projects that must be developed within a set of tight hardware, software, and operational constraints. $A = 3.6$, $B = 1.20$, $C = 2.5$, $D = 0.32$.

Levels above *basic* include factors for attributes of the product (e.g., required reliability), hardware (e.g., performance), personnel (e.g., software engineer capability), and project (e.g., required schedule). These factors have a multiplicative influence on the *development effort*. Values for the coefficients in the formulae are derived from experience with past software projects. It is an ongoing effort to collect data to be able to better calibrate future versions of the model. Current releases of a software for COCOMO claim to predict correctly within 20% of actuals around 60% of the time which is said to be a good value for effort prediction models. [Eng02] *COCOMO* has meanwhile been superseded by *COCOMO-II* which retains the fundamental approach, but introduces new factors and new values for model calibration.

2.2.2 Software design measurement

Design measurement is the application of measurement to design processes (all kinds of activities) and resulting design products (all kinds of documents). [Rom90] It offers advantages over more commonly used source code measurements. Measurement results are available in the design phase, not only at the back end of development, i.e., mainly coding and testing phases. Hence, it suggest a potentially high payoff since errors are assumed to be fixed more cheaply early in the software life cycle.

Recommendations outlined in [Rom90] for design measurement emphasize that there are many types of measurement goals, e.g., depending on object type or intended effect. Models and measures are seen as inseparable. Different types of measures are proposed, e.g., process and product, direct and indirect. Measurement-based analysis results should only be viewed as good as the data they are based on. A sound experimental approach is demanded, together with reporting measurement results in context, i.e., enabling repetition of an experiment; each experimental validation is to be assessed for transferable knowledge in future projects.

Two design steps are distinguished: *architectural*, or high-level, design and *algorithmic*, or low-level, design. Architectural design involves identifying software components and their interconnection. Algorithmic design involves identifying data structures and the control flow within the architectural components. Potential for architectural design measurement is limited by the measurability of design documents, or rather the lack thereof. This prompts the need for more formal and, hence, better measurable design processes.

2.2.3 Product quality

Software quality assurance collects fault data during phases of the software life-cycle, e.g. as number of faults divided by program size, number of customer change requests divided by program size.

Related to security faults, the total number of faults, the distribution of faults over a product's or version's lifetime is recorded. The time it takes for attackers to find a security hole and develop an exploit are related to a product's initial availability or to the time when knowledge of a security fault is made public. (cf., e.g., [Res04]) This can help system administrators to adapt deployment of software patches and to choose a software vendor with good reputation based on prior products. However, it does not help in assessing whether a given product is more secure than another, since the measurements are historic rather than predictive.

In addition, there are a couple of uncontrolled variables in these measurements, e.g., importance of a security fault, dependence on operating environment of the product, attacker behaviour.

2.2.4 Process quality

Software development *process measures* use finite sequential phases of development, and target control of resources. Resource usage and development costs are typically collected using corporate cost accounting systems. Examples include increase in program size per staff-day, percentage of completed planned work.

The *Systems Security Engineering Capability Maturity Model* (SSE-CMM) [SSE03] is a process reference model. It defines *what* an organization should do to achieve a secure software product; it does not define *how* the organization should do it and does not prescribe specific processes. Eleven process areas are identified that need to be scrutinized when developing secure software. These security best practices are:

1. *Administer security controls*: Ensure that intended security in system design is achieved in system in operational state.

2. *Assess impact*: Identify impacts of concern and likelihood of their occuring.

3. *Assess security risk*: Identify security risks based on threats, vulnerabilities, and impacts.

4. *Assess threat*: Identify security threats, their properties and characteristics.

5. *Assess vulnerability*: Identify and characterise system security vulnerabilities.

6. *Build assurance argument*: State assurance objectives supported by evidence.

7. *Coordinate security*: Assure that all parties are aware of and involved with security engineering activities.

8. *Monitor security posture*: Monitor and report all breaches of security, including attempted breaches and mistakes leading to possible breaches of security.

9. *Provide security input*: Transfer needed security information to system architects, designers etc.

10. *Specify security needs*: Identify needs to meet all legal, policy, and organizational requirements for security.

11. *Verify and validate security*: Verify and validate solutions against security requirements, architecture, and design using observation, demonstration, analysis, and testing.

For all activities in the process areas capability levels can be achieved. They indicate how well the activity is performed, whether it can be repeated in the same quality, and if it is kept track of and its performance improved by the organization. From low to high they range from:

1. *Performed informally*: Best practices of the process area are generally performed. The performance of these base practices may not be rigorously planned and tracked. Performance depends on individual knowledge and effort. Work products of the process area testify to their performance. Individuals within the organization recognize that an action should be performed, and there is general agreement that this action is performed as and when required. There are identifiable work products for the process.

2. *Planned and tracked*: Performance of the base processes in the process area is planned and tracked. Performance according to specified procedures is verified. Work products confirm to specified standards and requirements. Measurement is used to track process area performance, thus enabling the organization to manage its activities based on actual performance. The primary distinction from level one is that the performance of the process is planned and managed.

3. *Well defined*: Base practices are performed according to a well-defined process using approved, tailored versions of standard, documented processes. The primary distinction from level two is that the process is planned and managed using an organization-wide standard process.

4. *Quantitatively controlled*: Detailed measures of performance are collected and analysed. This leads to a quantitative understanding of process capability and an improved ability to predict performance. Performance is objectively managed, and the quality of work products is quantitatively known. The primary distinction from level three is that the defined process is quantitatively understood and controlled.

5. *Continuously improving*: Quantitative performance goals (targets) for process effectiveness and efficiency are established, based on the business goals of the organization. Continuous process improvement against these goals is enabled by quantitative feedback from performing the defined processes and from piloting innovative ideas and technologies. The primary distinction from level four is that the defined process and the standard process undergo continuous refinement and improvement, based on a quantitative understanding of the impact of changes to these processes.

SSE-CMM instructs developers to assess vulnerability of a product, but leaves it open to them to how to assess, and what framework and tools to use.

2.3 Security metrics

Research of security metrics is a rediscovered topic. In cryptography, for instance, strength of cryptographic algorithms has been examined for a long period. However, not always are a system's or a product's attributes as easy to distinguish as the length of a cryptographic key or the effort in man hours of a cryptographic brute force attack. Recently, there has been renewed interest in security metrics, e.g., in form of workshops like thoose organised by NIST CSSPAB in 2000 [Com00], ACSA in 2001 [AC01], and QoP 2005 [QoP05], 2006 [QoP06], and MetriCon 2006 [Met06].

In this section we discuss inter alia early indicators like *SECURATE* and *CCI*, *work factor*, *operating environment class*, and *attack surface* metrics. Setting up a metrics program without providing any concrete metrics is the topic of a NIST report on security metrics. Metrics have been introduced in and by security evaluation criteria, sometimes without explicitly mentioning these. We conclude with an overview of operational security metrics, i.e., assessments of the vulnerability of deployed systems, often based on known software versions for nodes in a network.

It appears that there are few product security metrics. Hence, there is a need for research in that area.

2.3.1 NIST (2003). Special publication 800-55

Report [NIS03] – *NIST SP 800-55* – introduces security metrics from a management perspective. Metrics are recommended as "tools designed to facilitate decision making and improve performance and accountability through collection, analysis, and reporting of relevant performance-related data." Different levels of maturity of a metrics program are defined, starting with a statement of goals and objectives, to measuring coverage of controls implementation, to testing and integration of detailed metrics.

From a technical point of view, detailed metrics at the highest levels of maturity are the most rewarding. Security metrics must yield quantifiable information for comparison purposes, apply formulae for analysis, and track changes using the same points of reference. NIST SP 800-55 mentions that percentages or averages are most common, and absolute numbers are sometimes useful, depending on the activity that is being measured.

As a help for security metrics documentation, a structured form is provided (cf. table 2.1). It helps evaluators and managers to understand metrics applied to the system under study. Categories in the form comprise desired results, actions to achieve the results, how to quantify completion of the objective, where to get the data upon which the measurements are based, and how to interpret numeric values and trends.

2.3.2 Security metrics in cryptography

In cryptography, some properties of algorithms can be regarded as security metrics. [MvOV96]

Key length, usually measured in *bits*, is the size of the parameter needed for encryption or decryption. A longer key typically means more effort for a computational brute force attack on the algorithm. Hence, the longer the key, the more secure an application of a cryptographic algorithm is. (On the other hand, a longer key also often involves more resources to encrypt or decrypt.) Key length is not an absolute metric. Different algorithms may operate with different key lengths while having the same effort for a

Category	*Content*
Performance Goal	Desired results
Performance Objective	Required actions to accomplish goal
Metric	Definition by quantitative measurements
Purpose	Insights to be gained
Implementation Evidence	Proof of controls existence
Frequency	Time periods for data collection to measure changes over time
Formula	Calculation to be performed; results in numeric expression
Data Source	Location of data to be used
Indicators	Meaning of metric and its performance trend

Table 2.1: Metric Detail Form according to NIST SP 800-55 [NIS03]

brute force attack. E.g., elliptic curve-based cryptography operates with shorter keys than traditional RSA public-key cryptography. However, the former is regarded equally secure with the functional advantage of employing a shorter key length. Key length is a good metric for different applications of the same algorithm. RSA with 4,096 bit keys is harder to attack than RSA with 1,024 bit keys.

The *work factor* is the minimum amount of work required to determine a cryptographic key. It can typically not be determined. The *historical work factor* is the minimum amount of work required to determine a cryptographic key using the best known algorithms at a given point in time. It can be used when searching for a lower bound on the *work factor*.

Cryptographic algorithms can be *empirically secure*, *provably secure*, or *unconditionally secure*. It is a measure of the assurance that an algorithm is secure against attacks. *Empirically secure* algorithms have been scrutinised by the scientific community for an extended period of time without someone being able to break them. It is comparable to the assurance provided by red team exercises in computer system security (cf. section 2.3.6). *Provably secure* algorithms have been mathematically proven to be as hard to attack as solving an associated mathematical problem. In addition, the difficulty of this underlying problem can induce an ordering on several provably secure algorithms. An *unconditionally secure* algorithm requires the attacker to know the cryptographic key to successfully attack the algorithm. At present, only the One-Time-Pad algorithm is known to possess this property.

More related to efficiency are measurements like *memory usage*, *speed*, *number of clock cycles per byte encrypted* (cf. e.g. [Gre01]).

2.3.3 Clements et al. (1977). SECURATE

SECURATE ([HMC78, Cle77]) is a computer installation security evaluation and analysis system. It is based on a model of a computer installation as a set of triples composed of objects, threats and security features. An evaluator assigns values to objects, likelihoods to threats, and effectiveness to security features. Assignments are valid for a triple only and can vary across triples, e.g. a security feature may be rated differently depending on object value and threat likelihood. Data items are given in terms of linguistic variables of a fuzzy rating language. This yields a scale with values *LOW*, *MEDIUM* and *HIGH*,

adjustable by *NOT*, *VERY*, *MOREORLESS*, *QUITE*, *PRETTY*, *SORTOF*, *REALLY*, *EXTREMELY*, *INDEED* and by conjunctions and relations. The linguistic variables are later mapped to compatibility functions represented by a vector in the APL programming language.

Measurements are done by human evaluators without tool support for automatic measurement and collection. All measurements are later entered into the SECURATE tool. It contains a model description of a typical computer system installation and allows modification of this default configuration. The largest installation evaluated by students as a term project comprises ca. 300 single ratings expressed in the rating language.

Evaluating security of a comprehensive system can be done by different evaluation functions (minimum, sub-set minimum, mean, mean weighted by value and threat likelihood, subsection mean weighted by average value):

- *Weakest Link* – this will look for the weakest feature resistance and return that as the security rating. The theory here is that the system is only as secure as its weakest link.

- *Selected Weakest Link* – this produces a weakest link rating based on those triples which satisfy the condition that either the object value or the threat likelihood is greater than a user specified minimum. The theory here is that one would only want to consider triples where the object is of at least a certain value or the threat is of at least a certain likelihood.

- *Fuzzy Mean* – this performs a fuzzy mean on the feature resistances and returns the result as the rating. The theory here is that a systems security is the mean of the security of its components.

- *Weighted Fuzzy Mean* – this performs as fuzzy mean on the feature resistance weighted by the greater of the object value and threat likelihood of each triple. The theory is that of the Fuzzy Mean, with the additional assumption that the more valuable objects and those with more likely threats should receive greater weight in the security rating.

- *Fuzzy Mean with Each Major Subsection Weighted by Maximum Object Value* – for each major subsection of the object specified, this finds the fuzzy mean of the resistances. It then weights these fuzzy means by the maximum object value found in the triples for each major subsection and averages these weighted means. The theory is similar to the Weighted Fuzzy Mean, but with the assumption that the major subsections should be weighted by their relative values, irrespective of the number of triples they each have.

A value for the overall security of a computer system installation could be given by SECURATE as e.g. "(MOREORLESS MEDIUM) TO (SORTOF HIGH), LOWEST RATING WAS GIVEN TO: OPERATING SYSTEM."

Reliability of this metric is not very high. It depends on the abilities of the evaluator, and different evaluators may arrive at different measurements, because there is no guidance in this phase.

Validity on the other hand is better. The five evaluation functions to compute a result based on measurements of components, reflect the traditional security perspectives.

Expressing the result in a rating language instead of providing a single number presumably leads to better understanding. However, empirical evidence of the usability part is rather anecdotal [HMC78].

2.3.4 Murine et al. (1984). Software Security Metrics

The article introduces the notion of *Software Security Metrics* (SSM), similar to the then examined *Software Quality Metrics*. It is a method to assure security of software systems by identifying and quantifiably measuring selected software criteria throughout the software development process.

Desirable software properties are called *factors*. Each factor is measurable by quantifying its constituents, called *criteria*. Individual *elements* of the criteria are derived from countable occurrences of the software attributes. A *count* is then called a *metric*. The elements themselves are objective (while their selection not necessarily is). Their metrics are often expressed as a ratio of compliances (successes) to occurrences (events). Different development phases may operate with a different selection of elements for given criteria.

Security criteria – on which security facors can be based – are shown in table 2.2 (taken from [MCC84], table 3).

Some security criteria are contrary to security quality criteria, e.g., *deceptiveness* vs. *self-descriptiveness*, or *security complexity* vs. *simplicity*.

Unfortunately, the method is explained only along a short example of ensuring integrity of a software system. It is unclear whether a complete set of security factors, criteria, and elements had been developed for the identified security factors.

2.3.5 Wood et al. (1987). Control Comprehensiveness Indicator/CCI

This work consists of a long checklist with hundreds of questions derived from literature, analysis of computer systems, and communication with experts on the subject of computer security. The authors also propose what they call a *Control Comprehensiveness Indicator (CCI)*. It is basically a weighted sum of questions on relevant implemented protective controls answered with "yes".

Each question, e.g. 2.6.16: "Are all changes in user privileges and in current passwords reflected in protected systems logs? (VH)," is assigned an importance on a scale of *very low*, *low*, *medium*, *high*, *very high*. These values can be adjusted with respect to the system under evaluation and may depend on system properties, protected assets, and perceived threats. Weights are attributed on a scale from zero to one according to the function in table 2.3.

Questions that are not applicable are not included in the further calculations. Questions answered with "yes" add the respective value to the achieved score, questions answered with "no" add zero. A maximum possible score is derived by adding all values under the assumption that all relevant questions are answered with "yes". The CCI is then given as the ratio of the achieved score and the maximum possible score.

Threat likelihoods or asset values are not included in the indicator, this is left to a quantitative risk analysis. The authors acknowledge that the CCI does not provide a means to measure an absolute level of security for a system. It might not even be possible to compare two systems unless they are very similar as regards controls implemented for

Criteria	*Definitions*
Access Audit	Those attributes of the software that provide for an audit of the access of software and data.
Access Control	Those attributes of the software that provide for control of the access of software and data.
Security Traceability	Those security requirements of the software system that provide a thread from the security requirements to the implementation with respect to software development and security environment.
Deceptiveness	Those attributes of the software that provide explanation of the implementation of a dissimilar function.
Simplicity	Those attributes of the software that provide implementation of functions in the most understandable manner. (Usually avoidance of practices which increase complexity.)
Security Complexity	Those attributes of the software that provide implementation of functions in the least understandable manner.
Inconsistency	Those attributes of the software that provide random design and implementation techniques and notation.
Perturbated Error Tolerance	Those attributes of the software that provide continuity of operation under randomly controlled conditions.
Security Completeness	Those attributes of the software that provide for full implementation of the security requirements.
Execution Efficiency	Those attributes of the software that provide for minimum processing time.
Storage Efficiency	Those attributes of the software that provide for minimum storage requirements during operation.

Table 2.2: SQM security criteria definitions

Importance	*Value*
Very low (VL)	0.1
Low (L)	0.3
Medium (M)	0.5
High (H)	0.7
Very high (VH)	0.9

Table 2.3: CCI weights for evaluation questions

protection. It is meant to give a coarse indication of general security problems (e.g., a CCI below 0.5) or of improvement of security posture of a system over time.

As a measuring technique, checklists and interviews are recommended. It is stated that "[a] number of other cost-effective ways to assess the adequacy of present controls [...] are available," alas this is not detailed.

More on the checklists can be found in appendix A.

2.3.6 Schudel et al. (2000). Adversary Work Factor

Schudel's approach [SW00] measures adversary work factor in man hours. Measurements are conducted by red team experiments. The work reports on a couple of red team experiments with the aim to support or refute some information security hypotheses, i.e., "Adding layers has at least a cumulative impact on adversary work factor" (sometimes), "Dynamic defense mechanisms can have a significant impact on adversary work factor" (accepted), and "[D]ynamic network reconfiguration effectively degrades the attacker's ability to map the network, and hence increases attacker work factor and improves system assurance" (refuted).

As regards measuring, focused experiments are recommended. It can be difficult to constrain even a cooperating adversary, so experiment setup should include realistic security mechanisms instead of hypothetical ones, where possible. Most information is seen to be gained from relative measures where one red team performs attacks with the same goals in different experiment setups. The authors warn that absolute values of adversary work factor contain little valuable information, owing to different behaviours, preparation, training, and talents of different red teams. They recommend establishing a baseline for red team and system performance, multiple runs of a given experiment, and limiting variables between each run to counter this variation in absolute values.

2.3.7 Bundesnetzagentur (2001/2005). Unified specification of operating conditions for signature creation applications

The regulatory body for electronic signatures in Germany issued a paper [Bun05] on operating conditions for signature creation applications. It is intended as guidance for developers and evaluators of software and seems to be the only structured attempt known so far to establish a *hierarchical ordering of the hostility of an operating environment.* Security requirements for signature creation applications are stated, derived from their legal basis, and a short list of possible threats is given. Security measures are to be applied either by the signature application component or the operating environment. Three classes of operating environments are distinguished: Unprotected working environment, Protected working environment, and Isolated working environment.

For a detailed discussion, see section 2.1.6.

2.3.8 Howard et al. (2003). Relative Attack Surfaces

In this article [HPW03] a metric with three dimensions is defined: *targets and enablers*, *channels and protocols*, and *access rights*. As a method of analysis a state machine model is proposed where the system and the adversary/threats are modeled by intended and actual states and transitions. A system under attack is then specified as $(System \bowtie Threat) \times Goal$, where *Goal* is a predicate over the state. An *execution* is an alternating

sequence of states and action executions, the *behaviour* is the set of all executions of a machine. A *vulnerable system* differs in actual behaviour from intended, i.e., by different states, initial states, actions, or transitions. An *attack* is a sequence of action executions, one of which involves an unintended state. It is distinguished between *process targets* and *data targets*. A target is the aim of an attack, an enabler is used during an attack. Channels are entry points to a system, e.g., sockets (message-passing type) or files (shared-memory type); protocols govern the rules of information exchange along channels.

Some Microsoft security bulletins are described in terms of this state machine model in the article, albeit a lot of natural English language is used. It is unclear how to use the description with tools for modelling and reasoning. The *attack surface* of a system is finally defined as a function from targets, enablers, channels, protocols, and access rights to presumably a single number or a triple. This (admittedly) simplistic definition of the attack surface function incurs the need for different functions that have to be determined by a security analyst for a given set of systems. Most likely, additional functions and weights will be used.

The state machine model is not used in the further presentation of an example Relative Attack Surface Quotient (RASQ) that one of the authors (Howard) used in an earlier MSDN (Microsoft Developer Network) publication. The authors mention problems with simple counting and comparison of different systems. They suggest to compare only different versions of the same system, e.g., under development or configuration.

2.3.9 Hunstad et al. (2004). A Method Based on Common Criteria's Security Functional Requirements

The approach presented in [HHA04] makes use of the CC [CC299b] to evaluate security of systems. *Security functional requirements* (SFRs) found in the CC are grouped into disjoint SFR components. For an evaluation, protection profiles are explored and relevant SFR components are determined. It is then assessed which SFR components are covered by the evaluated system components and each SFR component is assigned a value in the interval [0 . . 1] reflecting its security strength. When it comes to synthesis of results for separate components, a weight matrix can be used to prioritize some SFR components. Final ratings are given in the dimensions CIA (confidentiality, integrity, availability) and PDR (protect, detect, react). The idea is that values calculated for different systems help to compare overall security level of two systems and help to assess effectiveness of security improvements applied to a system. However, the authors state that evaluation of single components is "hard". They do not provide a method for this and assume that assignment of values is possible. It is emphasized that a system's components and relations have to be modelled, e.g. using UML, because SFR components can be effected by several system components.

2.3.10 Operational security metrics

Operational security metrics deal with concrete system deployments. They focus on known vulnerabilities in technical artefacts and on incorrect or insufficient configuration of components. The goal is to help system administrators decide which systems to patch or which systems have to be separated from others owing to possible attack paths. Operational security metrics are applicable in the deployment and operation phase of a software's lifecycle. They cannot be used during specification and implementation.

A couple of approaches use a graph representation or Petri nets for attack paths. [Dac94, DD94, SHJ+02] Data is typically obtained by repositories of known vulnerabilities [NJOJ03] and the patch status of nodes in a network, or it is gathered by *red team*-style experiments. [BOLJ94]

2.4 Metrics in security evaluation criteria

Evaluation of software products with respect to security started as an activity based on the personal experience of the evaluator. It still relies on that experience for a large portion. [KF05] To achieve a more reliable and repeatable process, evaluation criteria were developed. These standardise protection goals, recommend or request methods to check compliance of a target of evaluation with the goals, and provide levels for the description of attacker capabilities and the strength of protection mechanisms.

2.4.1 Trusted Computer System Evaluation Criteria (TCSEC)

The *TCSEC* [TCS85] are the first published comprehensive criteria for security evaluation of automatic data processing systems. They provide a hierarchy of seven (eight) classes, ordered by security effectiveness and assurance of evaluated systems. The criteria specify security mechanisms required for each class, as well as requirements to assurance, i.e., testing, verification, documentation.

Classes come in four divisions: D: *Minimal protection*, C: *Discretionary protection*, B: *Mandatory protection*, A: *Verified protection*. Characteristics of each class are presented in table 2.4. Higher divisions comprise all requirements of lower divisions.

Functional and assurance requirements are combined, i.e., to achieve a higher rating, both security mechanisms and assurance efforts have to be extended. From a procurement perspective this facilitates comparison of systems, since distinctions between classes are clear and their number is limited.

Most commercial operating systems achieve C2 level. Some systems at higher levels have been developed and are in use, albeit in niches.

2.4.2 German criteria for IT security (ITSK)

The early *German information technology security evaluation criteria* [Zen89] distinguish between evaluation of the *strength of security mechanisms* and *confidence in the correctness* of a system. Weaknesses of security mechanisms are discussed relative to the following aspects:

- Coverage of mechanisms to enforce the security policy
- Incorrect implementation
- Fundamental weaknesses even when implemented correctly

Fundamental weaknesses are further subdivided into weaknesses that can be reduced or eliminated by organizational means and inherent weaknesses that cannot be eliminated by organizational means or only with great difficulty. If they can be reduced by organizational means, this should be taken into account in the rating.
Six ordered levels of effectiveness of mechanisms are introduced:

Class name	*Security functional characteristics (cumulative)*
(Beyond A1)	(No functional enhancements specified, some suggestions for improvements in assurance given)
A1	functionally equivalent to B3
B3	Universally-invoked reference monitor, code not essential to security policy enforcement excluded from TCB, signalling of security-relevant events, trusted path isolated and distinguishable from others, trusted recovery
B2	Access control for all subjects and objects, covert channel analysis, structured TCB (Trusted computing base), principle of least privilege applied to TCB modules, trusted path for login and authentication
B1	Sensitivity labels for subjects and storage objects, label integrity upon export, MAC Mandatory access control
C2	Propagation of access rights limited, objects protected from unauthorized access, access control at single user granularity, freshness of resources, audit trail for object accesses
C1	Separation of users and data, cooperating users, single level of sensitivity, DAC Discretionary access control
D	Evaluated, but fails to meet requirements for higher evaluation classes

Table 2.4: Classes in the TCSEC

1. *Ineffective:* Mechanism not at all effective to prevent violations of the security policy.

2. *Weak:* Mechanism only suitable for preventing unintended violations of security policy.

3. *Moderate:* Protection against deliberate violations, can be overcome with moderate effort by persons familiar with the system.

4. *Strong:* Good protection against deliberate violations, can be overcome with great effort or with extensive insider support.

5. *Very strong:* Good protection against deliberate violations, can be overcome with great effort and with extensive support; if organizational measures are needed in addition, these have to be simple in design, have a low capability to error; error sources shall be monitored, error handling mechanisms have to be implemented by system.

6. *Virtually unbreakable:* Mechanism prevents all violations of security policy; according to present state of the art impossible to overcome; organizational measures only allowed if completely protected against errors by internal system monitoring functions.

There is no definition that distinguishes moderate effort from great effort. Apart from the highest level – *Virtually unbreakable* – the levels *Moderate*, *Strong*, *Very strong* can be associated with the three levels found in the ITSEC [ITS91] and Common Criteria [CC299a]; *Ineffective* and *Weak* then indicating a level below *Moderate* (ITSK [Zen89]) $\sim$ *Basic/Medium* (ITSEC [ITS91]) $\sim$ *Low* (CC [CC299a]).

The criteria provide general guidance to an evaluator how to examine security mechanisms and which questions to pose to a system under evaluation. However, only for authentication mechanisms specific directions are given to derive a mechanism's strength. When rating the guarantee of uniqueness of an identity, probability intervals are associated with an upper bound for ratings: Probability of an incorrect identification by a possibly ambiguous identity between 1 and 10^{-2} leads to a rating of *Weak*, 10^{-2}–10^{-4} *Moderate*, 10^{-4}–10^{-6} *Strong*, 10^{-6}–10^{-8} *Very strong*.

The accompanying evaluation manual [Zen90] contains exemplary guidance to assessment of security mechanisms. It focuses on use of [Zen89] and does not provide more specific advice as regards how specific vulnerabilities lead to a certain down-rating.

- A down-rating from *Virtually unbreakable* to *Very strong* can be done by showing that a higher level is not achievable.

- A down-rating from *Very strong* to *Strong* can be performed by showing that error monitoring and recovery are insufficient.

- Down-rating of authentication mechanisms happens by reference to probability intervals in [18].

- Rating of transmission errors is proposed by help of an unsupported table of probability intervals.

2.4.3 Information Technology Security Evaluation Criteria (IT-SEC)

Based on earlier national efforts in Germany, France, the U.K., and the Netherlands, the joint European criteria for information technology security evaluation [ITS91, ITS93] were developed. As in the German criteria [Zen89] – and contrary to the U.S.' TCSEC – a system's rating distinguishes between *assurance of correctness* and *strength of security mechanisms*.

A rating consists of two or three components. The first component is the *security target*, the template containing the security requirements that a *target of evaluation* (TOE) is evaluated against. The second component is a rating of the confidence one has in the correctness of the system, expressed as E0 to E6, with E6 being the highest level of confidence. A system that fails evaluation is awarded the lowest level: E0. In case of a successful evaluation a third component is used to indicate the strength of the security mechanisms employed. Strength is given as *basic*, *medium*, or *high*.
The criteria provide general guidance as to how strength of mechanisms is to be assigned:

- *Basic:* Mechanism provides protection against random accidental subversion, may be defeated by knowledgeable attackers.
- *Medium:* Mechanism provides protection against attackers with limited opportunities or resources.
- *High:* Mechanism could be defeated only by attackers possessing a high level of expertise, opportunity and resources; successful attack is judged to be beyond normal practicality.

Often only the E-level of confidence in the correctness is communicated in practice and used to compare different systems. Without stating the security target used as a baseline of the evaluation and without taking strength of mechanisms into account, the value of that approach is dubious.

2.4.4 Canadian Trusted Computer Product Evaluation Criteria (CTCPEC)

The *CTCPEC* [CTC93] were developed to provide a comparative scale for the evaluation of commercial products. They distinguish between functionality and assurance by independently and separately specifying security service requirements and assurance requirements. Functional criteria (confidentiality, integrity, availability, and accountability requirements) are composed of services, i.e., functional groupings to addres threats.
Each service contains levels. A level of service is a defined and measurable requirement for granularity or strength; as the level of service increases, a better defence against threats is provided. Levels of service are hierarchical in terms of protection but are not necessarily proper subsets in all cases.

As an example the Trusted Path criteria (WT) in the Accountability section (W) is discussed. WT-0: A level of 0 always stands for non-compliance with respect to a functional requirement, i.e., no trusted path is present. WT-1: Trusted path for initial identification and authentication, trusted path communication exclusively initiated by user. WT-2: Trusted path for initial identification and authentication and at other times

when direct communication between user and TCB is needed, communication initiated by TCB must be identifiable and requires positive user confirmation. WT-3: Trusted path as defined by WT-2, but communication between user and TCB must always be secured in both directions.

A rating is given as a set of functionality and assurance levels, e.g. as "CD-2, CR-1, AC-1, WI-1, WA-2, IS-1, T-2". This example rating requires the product to fulfil Basic Discretionary Confidentiality (discretionary confidentiality level 2), Object Reuse, Quotas (containment availability level 1), External Identification and Authentication (I&A accountability level 1), Security Audit (audit accountability level 2), Basic Separation of Duties (separation of duties integrity level 1), Assurance Level 2.
To facilitate a mapping from TCSEC evaluations to CTCPEC evaluations, four security functionality profiles are specified to reflect TCSEC C2, B1, B2, B3 functional security requirements respectively.

In contrast to the TCSEC, the CTCPEC treats users and processes differently. The TCSEC notion of an active subject is split up into a user and a process. Hence, mediation of accesses can be based on security attributes of both user and process. Everything is treated as a passive, user, or process object depending on their role in an interaction.

2.4.5 Common Criteria (CC) and Common Evaluation Methodology (CEM)

The approach of the Common Criteria [CC299a, CC299b, CC299c] is to present an internationally accepted framework for evaluation of IT security. It is divided into three parts: the first part introduces the general model, the second concentrates on the *security functional requirements*, and the third addresses *assurance requirements*.

The idea is that consumers specify their security needs in implementation-independent protection profiles. Vendors can then develop products that meet these needs. A security target is the implementation-specific definition of a product's security requirements. It can be subjected to evaluation by an independent body. Evaluators then check the provided documents and perform tests on the product to conclude whether the target of evaluation fulfils the desired security goals at the desired level of assurance.

Evaluation results basically consist of two pieces of information: the *protection profile* or *security target*, and the *evaluation assurance level*, given as EAL1 (functionally tested) to EAL7 (formally verified design and tested). Often, only the evaluation assurance level (EAL) is provided when referring to an evaluated product. That omits what the evaluation comprised, as protection profiles can greatly differ in requirements. The EAL reflects quality and thoroughness of the evaluation process while the protection profile defines security goals and protection mechanisms to address threats.

The result of an evaluation is binary – the evaluated product either meets the security requirements or fails. An evaluation fails because a security mechanism is too weak or because advanced methods on higher EAL's fail to confirm correctness of a security mechanism. In the latter case, the product might still meet the defined security requirements even if it was not possible to confirm that fact with the accuracy asked for.

Levels of security are introduced by having three classes of attackers with *low*, *moderate* or *high* attack potential, respectively. Attack potential is defined as "[t]he perceived potential for success of an attack, should an attack be launched, expressed in terms of an attackers expertise, resources and motivation." However, this concept is not elaborated

Factor	*Range*	*Identifying value*	*Exploiting value*
Elapsed Time	< 0.5 hour	0	0
	< 1 day	2	3
	< 1 month	3	5
	> 1 month	5	8
	Not practical	* (practically not exploitable)	* (practically not exploitable)
Expertise	Layman	0	0
	Proficient	2	2
	Expert	5	4
Knowledge of TOE	None	0	0
	Public	2	2
	Sensitive	5	4
Access to TOE	< 0.5 hour, or access undetectable	0	0
	< 1 day	2	4
	< 1 month	3	6
	> 1 month	4	9
	Not practical	* (practically not exploitable)	* (practically not exploitable)
Equipment	None	0	0
	Standard	1	2
	Specialised	3	4
	Bespoke	5	6

Table 2.5: Calculation of attack potential (based on table [CEM04] B.8.3)

further. In the context of strength of mechanisms low attack potential is associated with casual breaches of security, moderate with straightforward or intentional breaches, and high attack potential with deliberately planned or organised breaches. The Common Evaluation Methodology [CEM04] used to guide the work of the evaluator gives some hints towards attack potential. The CEM does not define what *low*, *moderate* or *high* attack potential is. It instead provides a table with points attributed to properties and requirements of an attack. The points are summed up and then translated to attack potential depending on the interval they belong to. The table and intervals are reproduced in table 2.5 and table 2.6. In table 2.5 there are points for identifying a vulnerability as well as for exploiting it. The rationale is that there could be some vulnerabilities that are hard to find and then easy to exploit, or easy to find but hard to exploit.

Points for the five factors in columns *Identifying value* and *Exploiting value* are summed up. The sum of the ten values is then used with table 2.6 to determine the class of attack potential. The calculation should help the evaluator in determining the appropriate attack potential and should provide a common basis.

In terms of metrics, reliability of this metric is estimated to be high. As regards validity of the results, there exists only anecdotal evidence. [KF05] The tables are supposed to have originated from a set of attacks deemed to be representative. Values for each factor have been adjusted so that the experts present at time of decision could agree that results of the calculations matched their intuitive assessment of attack potential. Unfortunately,

Range of values	*Resistant to attacker with attack potential of*
< 10	No rating
$10 - 17$	Low
$18 - 24$	Moderate
> 24	High

Table 2.6: Rating of vulnerabilities (based on table [CEM04] B.8.4)

EAL	*Component*	*Protection against attacker with attack potential of*	*Insufficient protection against attacker with attack potential of*
EAL1	–	n/a (not applicable)	n/a
EAL2	AVA_VLA.1	Only identification of vulnerabilities by developer	n/a
EAL3	AVA_VLA.1	Only identification of vulnerabilities by developer	n/a
EAL4	AVA_VLA.2	Low	Moderate
EAL5	AVA_VLA.3	Moderate	High
EAL6	AVA_VLA.4	High	n/a (successful attack beyond practicality)
EAL7	AVA_VLA.4	High	n/a (successful attack beyond practicality)

Table 2.7: EALs and attack potential (based on tables [CC299c] B.19 and [CEM04] B.8.1, B.8.2

this set of representative attacks is not documented and the results can not be verified without.

Interestingly, deviations from this method have to be justified by an evaluator; [CEM04] states in B.8.2 (paragraph 1823): This approach should be adopted unless the evaluator determines that it is inappropriate, in which case a rationale is required to justify the validity of the alternative approach.

Most evaluations of commercial products are completed with EAL4 or lower. This includes only certified resistance against vulnerabilities with low attack potential. In table 2.7 *AVA_VLA* stands for the component *Vulnerability Assessment* of the CC.

The CC and CEM in versions prior to 3.0 ([CC299a, CC299b, CC299c, CEM04]) distinguish between *strength of security mechanisms* and *assessment of vulnerabilities*. Strength of mechanisms is analysed for all permutational or probabilistic mechanisms. This class of mechanisms can be attacked directly (without bypassing, tampering with, or misusing a mechanism), regardless of the quality of the implementation. In practice, this class comprises password mechanisms, hash functions, and biometric authentication. [KF05] Cryptographic mechanisms have always been judged separately.

Starting with the revised version 3.0 ([CC305a, CC305b, CC305c, CEM05]), however, this distinction between strength of mechanism and assessment of vulnerabilities is no longer made. Evaluations performed against earlier versions of the CC have shown that the

EAL	*Component*	*Protection against attacker with attack potential of*	*Insufficient protection against attacker with attack potential of*
EAL1	AVA_VAN.1	Basic	Extended-Basic
EAL2	AVA_VAN.2	Basic	Extended-Basic
EAL3	AVA_VAN.2	Basic	Extended-Basic
EAL4	AVA_VAN.3	Extended-Basic	Moderate
EAL5	AVA_VAN.4	Moderate	High
EAL6	AVA_VAN.5	High	Infeasible
EAL7	AVA_VAN.5	High	Infeasible

Table 2.8: EALs and attack potential (based on tables [CC305c] E.24 and [CEM05] B.3

distinction was questionable when assessing products. Hence, evaluation of the strength of mechanisms is proposed to become a part of vulnerability analysis. Calculation of attack potential is adjusted somewhat. New levels of attack potential are introduced – *basic* and *extended-basic* – that replace the previous level of *low*. The factors to include in an analysis stay the same. Hence, the fundamental concerns regarding this approach still hold. The new requirements for attack potential at a certain level of assurance are shown in table 2.8.

Part of a CC evaluation is the analysis of the architectural design of a target of evaluation. This can be done either at high level (major structural units, i.e., subsystems) or low level (including the internal workings). The criteria request only that the presentation of the design should be informal, semi-formal, or formal, and the level of detail that has to be documented. They do not provide an explicit model to use.

This lack of an explicit model in the CC is also pointed out by [Whi01], stating "[t]o develop an extensible method for designing secure solutions, additional work is required to develop: 1. A system model that is representative of the functional aspects of security within complex solutions. 2. A systematic approach for creating security architectures based on the Common Criteria requirements taxonomy and the corresponding security system model."

2.5 System specification

We survey three different methods of formally specifying a software systems. *UML* (Unified Modeling Language) is an extensibe graphical notation for the specification of software-intensive systems at varying levels of detail; it that has found broad acceptance in industry. *Data spaces* are based on a significantly more rigorous mathematical foundation than *UML*. However, tool support for this technique is next to non-existent. *Z* is a standardised specification notation with a mathematical basis in set theory; tool support is good.

2.5.1 UML and Patterns

Patterns for security [FP01] are templates for software developers to integrate security mechanisms, e.g., authorization, role-based access control, and multilevel security. They

are relatively high-level and are not related to specific attacks. There exists some work [YJB97] on patterns for architectural structures, namely Secure Access Layer, Single Access Point, Check Point, Roles, Session, Limited View, and Full View With Errors.

A recent article [BWG05] – using rather uncommon terms with respect to security research – discusses the use of security patterns to meet security objectives. Four architectural security properties are identified: Error management, Simplicity, Access Control, and Defense in depth. These are linked to security objectives (e.g., confidentiality, integrity, availability) and to security patterns (cf. [YJB97]). Patterns guide implementation of security properties which in turn fulfil security objectives. The approach appears fruitful, however, the architectural security properties seem quite randomly selected.

Similar to familiar use-case diagrams, specification of misuse cases is proposed by [SO01]. They provide a method to include misuse/attacks in the software design process and to lift awareness of security problems. However, they are rather high-level and lack consistent semantics. Knowledge of attacks has to be brought in externally.

CORAS [COR04] offers a risk analysis and modeling language for information security. A specialization of UML is used and based on threats, vulnerabilities, unwanted incidents, risks, and treatments. Knowledge of vulnerabilities and possible attacks is supposed to be gathered during the risk analysis phase. Description of scenarios can vary according to the level of detail a developer is willing to present.

Donner [Don03] emphasizes the need for an ontology of the security field, i.e., a set of descriptions of the most important concepts and the relationships among them. This would be especially helpful when analysing attacks and classifying them with other similar things. A security ontology in the context of web services security, supporting credentials and communication protocols, is proposed by [DKF+03]. Information flow and access control (here under the terms of secure entities and dependencies) are included in an ontology for a framework for multi-agent systems. [MGM03]

2.5.2 Data spaces

A *data space* $\mathfrak{D}$ is a triple $(X, \mathfrak{F}, p)$, consisting of a *state space* X, a *processor* p, and a set $\mathfrak{F}$ of functions with a common domain X and whose ranges are value sets of data types. [CH78a, CH78b] The set $\mathfrak{F}$ can be interpreted as a state description, assigning values to a state object. One could, e.g., have functions like a *program counter* or *registers* to model a common central processing unit's architecture.

Two important characteristics for state structures of a virtual machine are named: *completeness*, i.e., states determine all their descriptor values in a unique way and states are described by their complete set of descriptor values, and *orthogonality*, i.e., descriptors do not depend on each other. Both are properties of the function set $\mathfrak{F}$.

Definition: A set $\mathfrak{F}$ of functions with common domain X is *complete* for X if, for all $x,\ y \in X$, $f(x) = f(y)$ for all $f \in \mathfrak{F}$ implies that $x = y$.

Definition: A set $\mathfrak{F}$ of functions with common domain X is *orthogonal* (for X) if, for every function $\eta\ :\ \mathfrak{F} \rightarrow X$, there exists z in X such that for all f in $\mathfrak{F}$ we have $f(z) = f(\eta\,(f))$.

Some examples for data spaces representing simple programs in different program languages are given in the original paper [CH78a]. This approach of an executable formal specification had probably been presented ahead of time, since it did not lead to immediate broad follow-up research by the community. Similar ideas were later rediscovered by other researchers.

More recently, data spaces have been employed to describe the FlexiBeans component model. [Sti00]

2.5.3 Formal specification in Z

The formal specification notation Z is based on Zermelo-Fraenkel set theory and first order predicate logic (cf. e.g. [Spi92, Jac97, PST96, BSC94, Bow96]). It has been developed by the Programming Research Group (PRG) at the Oxford University Computing Laboratory (OUCL) and elsewhere since the late 1970s. It is now defined by an ISO standard. [Z00]

[ASP03] presents templates to translate UML class diagrams to Z specifications. [KC00] discusses a method to define UML models as Object-Z specifications (Object-Z is an object-oriented enhancement to Z).

A comprehensive discussion of formalised architecture description and analysis is given in [AAG95]. An architecture is modeled as components and connectors. Architectural styles are defined by meaning functions assigning an interpretation to components and connectors. The article presents examples of pipes and filters, and event systems. Properties of these styles (e.g. composability) are proved. Z is used throughout the article.

Z is also used by [SSM97] to reason about an architectural standard, in this case COM (Component Object Model). Formalisation helped to prove that a desired software architecture would not have been possible, i.e. violated the standard. Some properties of COM that were not specified explicitly became clearer during the process.

Definition and refinement of a "Message Router" architectural style is shown in [CM97]. Properties of the style, e.g. order preserving, are proved. The authors use a chemical abstract machine model to analyse dynamic properties of this non-sequential system. In [CS00] it is argued that combining a traces-based model with a state+operations style Z specification is tedious. Z offers better tools support, so the additional effort appears justified.

Z is used as the basis for different representations, i.e. natural English, Prolog, ad-hoc graphical notation, Petri nets by [MS90]. It is claimed that different presentations cater better to the needs of different audiences and increase understanding. Consistency is found to be difficult without adequate tool support. Not one representation is found to be "best". Criteria for good specifications are given in [Gra90]. Hints are provided with respect to naming, state vs. operations, abstraction, use of implications, and close correspondence between mathematical definitions and the accompanying readable description.

Chapter 3

Software metrics for resilience

Chapter summary: In this chapter we show how security of programs against malware attacks can be ranked by assigning a resistance class to a program. A resistance class is composed of a score for security requirements and a score for attacker capabilities. Security requirements are expressed by three scales: *Data integrity*, *Data confidentiality*, and *Code integrity*. Attacker capabilities are expressed by four scales: *Attack initiation capability*, *Available time to attack*, *Attack variation capability*, and *Influence on the local human user*. Measuring resistance of a program is performed by fourteen software security metrics. These measure compliance of the program's architecture with established design principles for secure programs.

Determining the level of resilience a software product has against malware attacks is important. Presence of malware on a computer is a real scenario (cf. spyware, viruses) [Mye80], albeit being regularly neglected when evaluating products. The user often has to ensure by organisational or other means, e.g. by installing antivirus software, that the environment of the target of evaluation is free of malicious software. Otherwise, the evaluation result is not valid and the evaluating body does not make any assumptions about the software operating as expected. [Lan06b]

Evaluation, e.g. following the Common Criteria process, is time consuming, and is often begun after product implementation. When it has been completed, the evaluated version may already be functionally outdated. We want a measure based on an application's architectural description. This is available after the design process and allows an early estimation of resistance against malware attacks.

To do this, we describe the system architecture, attackers' capabilities, and attacks by generic *building blocks*. Common security requirements and the desired level of resistance are defined by *resistance classes*. A class contains all architectures that are resistant to attacks with a certain potential. It can then be determined whether a concrete system architecture is a member of a certain resistance class. By establishing an order on security requirements and attacker capabilities, resistance classes can be ordered. A system architecture in a higher class can then be regarded as more secure – in the presence of malware attacks – than an architecture in a lower class.

3.1 Ranking of security requirements

Security is traditionally viewed as a binary property – a product either is secure or it fails to fulfill its security requirements. Still, there is a significant variation in the possible requirements. One product can live up to more stringent requirements than another, hence, be a *more secure* product. This suggests an ordering of security requirements.

It is noted by [LABMC94] that "[a] security flaw is a part of a program that can cause the system to violate its security requirements. These requirements vary according to the system and its application." We assume a set of typical security requirements shared by our class of security-sensitive programs. Without a definition of security requirements, no measurement of goal completion would be possible since security would be undefined.

We are concerned with local security-sensitive programs. Even if typical access control systems associate processes with the users who created them, not all processes of the same user are equal. Software may be used as an intermediary to other objects or services, i.e., it brokers requests as a *Protected Subsystem*, cf. [SS75], or *Unix*-style `suid` programs. Storage controlled by a process may contain security-relevant data, e.g., credentials or copyright-restricted material received from a remote service.

Definition 3.1.1. *A* local security-sensitive program *p is a program being executed on a personal computer or similar device, performing a security-sensitive function, and being used by a local human user. Security-sensitive functions include, but are not limited to, protecting the confidentiality and integrity of stored data items, modifying data items according to the intentions of the user, and reliably executing its code.*

Hence, code and all data controlling its execution must be protected against modification, replacement, or subversion. *Protected* in this respect means to *prevent* or *detect* unauthorized access. Unauthorized actions are security-relevant actions occuring contrary to the desire of the person controlling an asset.

An attacker does not have direct physical access and attacks are carried out by other processes on the device. The security requirements state which goals must not be violated for the system to continue being regarded as *secure*. In our scenario we have the following security requirements: *data integrity* and *data confidentiality*, as well as *code integrity*. Data items may contain sensitive information or information that must not be manipulated. Code integrity covers the correct performance of mechanisms. These may operate on temporary data items or control valuable activity that does not lead to stored data items, e.g., moving a robot actuator.

Our focus on integrity (of data and code) as the primary security requirements and on confidentiality as secondary is supported by the observation of real threats in recent years, cf. [MY06].

Security requirements can hence be described by three axes (data integrity, data confidentiality, code integrity). Each axis is assigned a value on the following scale:

1. Attack remains undetected

2. Attack is logged and detectable (on request)

3. Attack is detected and the user is alerted (when it happens)

4. Attack is prevented

A low value stands for a weak security requirement. A high value implies a strong protection goal.

Lemma 3.1.1. *A system that logs a successful attack on it is more secure than a system that does not detect an attack.*

A system that detects an attack and alerts the local human user is more secure than a system that logs an attack, but does not alert the user.

A system that prevents an attack is more secure than a system that detects and attack, but does not prevent it.

Proof 3.1.1 (Lemma 3.1.1). *Omitted.* □

Applying the lemma, the scales for the three requirements data integrity, data confidentiality, code integrity are interpreted as follows:

- Limit damage done to data integrity
  1. Modification of data components remains undetected
  2. Modification of data components is logged and detectable (on request)
  3. Modification of data components is detected and the user alerted (when the attack happens)
  4. Modification of data components is prevented
- Limit damage done to data confidentiality
  1. Disclosure from data components remains undetected
  2. Disclosure from data components is logged and detectable (on request)
  3. Disclosure from data components is detected and the user alerted (when the attack happens)
  4. Disclosure from data components is prevented
- Limit damage done to code integrity
  1. Modification of code components remains undetected
  2. Modification of code components is logged and detectable (on request)
  3. Modification of code components is detected and the user alerted (when the attack happens)
  4. Modification of code components is prevented

Mathematically, security requirements of a local security-sensitive program are defined as a triple:

Definition 3.1.2. *Let $S \subseteq \mathbb{N}$ be the* strength *of a security requirement. The set of security requirement levels is then defined as $SecReqLevel = S \times S \times S$. (*=data integrity $\times$ data confidentiality $\times$ code integrity*)*

Definition 3.1.3. *A security requirement level* $s_1 = (di_1, dc_1, ci_1) \in SecReqLevel$ *is* equivalent *to a security requirement level* $s_2 = (di_2, dc_2, ci_2) \in SecReqLevel \Leftrightarrow di_1 = di_2 \wedge dc_1 = dc_2 \wedge ci_1 = ci_2$. *We write* $s_1 = s_2$.

A security requirement level $s_1 = (di_1, dc_1, ci_1) \in SecReqLevel$ *is* at least as strong *as a security requirement level* $s_2 = (di_2, dc_2, ci_2) \in SecReqLevel \Leftrightarrow di_1 \geq di_2 \wedge dc_1 \geq dc_2 \wedge ci_1 \geq ci_2$. *We write* $s_1 \geq s_2$.

A security requirement level $s_1 = (di_1, dc_1, ci_1) \in SecReqLevel$ *is* at most as strong *as a security requirement level* $s_2 = (di_2, dc_2, ci_2) \in SecReqLevel \Leftrightarrow di_1 \leq di_2 \wedge dc_1 \leq dc_2 \wedge ci_1 \leq ci_2$. *We write* $s_1 \leq s_2$.

Definition 3.1.4. *A security requirement level* s_1 *is* stronger *than a security requirement level* s_2 *if* s_1 *is at least as strong as* s_2 *and* s_1 *is not equivalent to* s_2. *We write* $s_1 \gneq s_2$.

Weak requirements are characterised by low values for the components of their associated security requirement levels. Strong requirements are characterised by high values for the components of their associated security requirement levels.

3.2 Attacker capability metrics

A process (i.e., the attacker's tool) has some inherent capabilities depending on the system it inhabits. These generally include access to securable objects, e.g. files, according to the access control configuration. (Contemporary discretionary access control systems assign the same privileges to all processes bound to the same user account.)

In addition to these, attacker-specific capabilities of a malicious process can be described along four axes: *attack initiation capability*, *available time to attack*, *attack variation*, *influence on user*:

Definition 3.2.1. Attack initiation capability *is the attacker's capability to control when an attack is begun and whether an attack can be attempted repeatedly. Possible values in ascending strength are* Initiation by user action*,* Initiation by system action (automatic)*,* Initiation at attacker's discretion*.*

Definition 3.2.2. Available time to attack *is the attacker's capability to attempt an attack over time. Possible values in ascending strength are* Once per user session*,* Several times per user session*,* Throughout whole user session*.*

Available time to attack must not be confused with the *window of opportunity*. The window of opportunity describes the time/interval in which an attack is possible and depends on the system under attack. *Available time to attack* is based on the attacker's abilities to launch an attack within this window of opportunity.

Definition 3.2.3. Attack variation *is the attacker's capability to adjust an attack while the attack is in progress. Possible values in ascending strength are* Attacker cannot customise an attack*,* Attacker can customise automatically apart from passing a Turing test*,* Attacker can customise automatically including passing a Turing test*.*

Attack variation is an indication of the flexibility of the attacking process to adapt the attack to changes in the attacked application's modules or (user) interface. An attacking process being able to pass a Turing test could e.g. be one with a real-time communication channel to a human accomplice.

Definition 3.2.4. Influence on user *is the influence that the attacker can exercise on the local human user. Possible values are* User cannot be influenced*,* User can be influenced once during attack*,* User can be influenced repeatedly during attack*.*

It might be argued that the user should be part of the system under attack. However, the user is part of the application's architecture only insofar as the user is concernced as a source of input or destination to receive output. An assessment of an application's resilience against malware attacks should be independent from the user's ability to withstand *social engineering* techniques. Incorporating this ability as the attacker's capability to influence the user allows the rating of the system to remain constant. Raising security awareness among users, for instance, does not change an application's resilience. The attacker's capabilities, however, depend on the user's susceptibility to social engineering.

Another option might be to remove dependability on the user from the description of an attacker's capabilities. We would then operate with an application's resilience (constant if application is not changed), attacker's capabilities (constant if attacker is not changed), and a user's resilience to social engineering (constant if user is not changed). Our compound attacker capabilities rating can be used to that purpose by projecting its first three components as *pure attacker capabilities* and projecting the fourth component as *user susceptibility.* We stick to the compound attacker capabilities rating for the remainder of this thesis.

For the four attacker capabilities attack initiation, available time to attack, attack variation, influence on user the scales are interpreted as follows:

- Attack initiation capability
  1. Attack can be initiated by user action
  2. Attack can be initiated automatically
  3. Attacker can launch attack at own discretion
- Available time (*time to carry out an attack*; in contrast to *time to intrusion* which is not applicable – attacks typically fail or succeed in an instant)
  1. Attacker can attempt attack only once per user session (e.g. post-condition of attempted attack invalidates its pre-condition)
  2. Attacker can attempt attacks several times during user session (e.g. when guessing a password)
  3. Attacker can carry out attack as long as user session lasts
- Attack variation
  1. Attacker cannot customise attack, follows same procedure every time (e.g. uses same script)
  2. Attacker can customise attack automatically, but cannot pass Turing test
  3. Attacker can customise attack interactively (process has access to human collaborator), can pass Turing test (e.g. can solve CAPTCHA – Completely Automated Public Turing test to tell Computers and Humans Apart [vABHL03])
- Influence on user (e.g. deception via user interface manipulation, user activates components that are then attacked)

1. User could neither be enticed nor forced to act
2. User could be enticed or forced to act once
3. User could be enticed or forced to act repeatedly in similar manner

Mathematically, capabilities of an attacker using a locally executed process as a tool are defined as a quadruple:

Definition 3.2.5. *Let* $C \subseteq \mathbb{N}$ *be the* strength *of an attacker's capability. The set of attacker capability levels is then defined as* $AttCapLevel = C \times C \times C \times C$.

Definition 3.2.6. *An attacker capability level* $c_1 = (ai_1, at_1, av_1, ui_1) \in AttCapLevel$ *is* equivalent *to an attacker capability level* $s_2 = (ai_2, at_2, av_2, ui_2) \in AttCapLevel \Leftrightarrow ai_1 = ai_2 \wedge at_1 = at_2 \wedge av_1 = av_2 \wedge ui_1 = ui_2$. *We write* $c_1 = c_2$.

An attacker capability level $c_1 = (ai_1, at_1, av_1, ui_1) \in AttCapLevel$ *is* at least as strong *as an attacker capability level* $s_2 = (ai_2, at_2, av_2, ui_2) \in AttCapLevel \Leftrightarrow ai_1 \geq ai_2 \wedge at_1 \geq at_2 \wedge av_1 \geq av_2 \wedge ui_1 \geq ui_2$. *We write* $c_1 \geq c_2$.

An attacker capability level $c_1 = (ai_1, at_1, av_1, ui_1) \in AttCapLevel$ *is* as most as strong *as an attacker capability level* $s_2 = (ai_2, at_2, av_2, ui_2) \in AttCapLevel \Leftrightarrow ai_1 \leq ai_2 \wedge at_1 \leq at_2 \wedge av_1 \leq av_2 \wedge ui_1 \leq ui_2$. *We write* $c_1 \leq c_2$.

Definition 3.2.7. *An attacker capability level* c_1 *is* stronger *than an attacker capability level* c_2 *if* c_1 *is at least as strong as* c_2 *and* c_1 *is not equivalent to* c_2. *We write* $c_1 \gneq c_2$.

Weak attackers are characterised by low values for the components of their associated attacker capability levels. Strong attackers are characterised by high values for the components of their associated attacker capability levels.

3.3 Generic attacks

After having defined *SecReqLevel* and *AttCapLevel* we want to define *resistance classes*. For this, we first need a repository of attacks, *AttRepository*, and a function mapping attacks to the capabilities an attacker must possess to perform them, *NecAttCap*.

Malware intends to violate the protection goals of another process. To this end, it can modify data items directly. If data items are protected, then an attack must focus on processes that have access to the protected data items. These processes can be influenced in one of three ways: directly, by modifying the parameters for the code, by influencing the user. We summarise:

- Malware can *directly attack* integrity and confidentiality of data items directly, i.e., without interfering with other processes.
- Malware can attack the *integrity of the code* that is executed. The behaviour of the code is hence changed directly.
- If integrity of the code is preserved, then malware can attack the *integrity of the stored or transmitted parameters* the code uses to determine its execution path. The behaviour of the code is hence changed depending on the values of the parameters.
- If integrity of the code and its parameters is preserved, then malware can *influence the operator* (local human user) to misuse the process. The behaviour of the code is hence changed according to user actions.

Table 3.1: Categories for generic attacks (cf. [CEM04])

Category	*Method*
Accessing stored data items directly	– Exploit weak access control configuration
Violating code execution integrity	– Exploiting absence of security enforcement on interfaces, e.g. replacing internal modules – Access to unprotected shared objects, e.g. replacing or modifying shared modules
Changing parameters	– Exploiting absence of security enforcement on interfaces, e.g. modifying scripts, configuration data – Executing additional or unintended components, e.g. loading run-time libraries, inheriting privileges – Using components in unexpected context or for unexpected purpose, e.g. simulating user actions – Access to unprotected shared objects, e.g. input to process – Modifying a component's environment, e.g. configuration data – Causing extreme circumstances for components, e.g. exhausting shared resources
Influencing the user	– Access to unprotected shared objects that comprise the user interface, e.g. manipulating shared desktop area

If executed code and parameters are unchanged and if the process is operated correctly by the user, it produces the intended results. No manipulation is possible. Of course, at a lower level of execution, e.g. by a manipulated virtual machine, an attack might be possible. Attacks at a lower architectural level are outside the scope of this work. We discuss attacks by malicious processes running at the same level of abstraction as the attacked processes.

Generic attack methods to fill these four classes – direct access to data items, violating code integrity, violating parameter integrity, influencing the user – can, among other places, be found in the evaluation manual [CEM04] accompanying the Common Criteria. Here, attacks are classified into *bypassing of security enforcement*, *tampering with security mechanisms*, *direct attacks on permutational* (e.g. hash function) or *probabilistic* (e.g. password, biometric) security mechanisms, or *misuse of systems* (cf. section AVA_VLA of [CEM04]). With respect to architectural security, this provides us with the repository of typical attacks shown in table 3.1.

We identify 13 generic attack methods amenable to malware. These are listed in table 3.2 and formally described in section 6.1.

This list of generic malware attacks is constructed with the concept of a Turing machine in mind. Turing machines have been used in malware analysis earlier (cf. [Coh85], [Lei00]). Consider a Turing machine (Q, Σ, δ, q_0) with a set of states Q, a set of tape symbols Σ, a transition function δ, and a start state q_0. An attacking malicious process can read and modify the output on the tape (i.e. the stored data items), can modify the transition function δ (i.e. the executable code), can read and modify the input on the tape (i.e. the parameters), can modify the start state q_0 (i.e. influence the user operating the machine). Hence, our categorised list of generic attacks comprises all targets accessible to malware.

Table 3.2: Generic malware attack methods

Category	*Method*
Directly violate integrity, confidentiality of stored data	– Modify stored data item – Retrieve contents of stored data item
Violate integrity of executed code	– Modify code in memory – Modify stored code module – Add stored code module – Modify reference to stored code module
Violate integrity of parameters	– Initiate communication with component and send data – Respond to component's communication request – Modify stored data item containing parameters – Modify reference to stored data item – Simulate user input
Influence user	– Modify user interface object – Modify stored data item examined by users for decisions

For each attack in the set of all generic attacks, *AttRepository*, the necessary capabilities are recorded by $\mathit{NecAttCap} : \mathit{AttRepository} \rightarrow \mathit{AttCapLevel}$. If a specific attack can be applied in a certain system configuration depends on the attack's preconditions. We take *NecAttCap* for given for the moment. It is defined in section 6.3.1 in the context of the formal description of generic attack methods.

3.4 Resistance classes

Metrics for security requirements and metrics for attacker capabilities lead to the definition of resistance classes. A *resistance class* is defined as the set of all system states in which an attacker with given capabilities cannot violate the given security requirements. Let *Conf* be the set of all system configurations.

Definition 3.4.1. *The* EvalState *function maps a system state to the highest security requirements level that is possible for this state.*

$$\mathit{EvalState} : \mathit{Conf} \rightarrow \mathit{SecReqLevel}$$

The ApplyAttack *function maps a system state and an attack to the resulting after-attack state.*

$$\mathit{ApplyAttack} : \mathit{Conf} \times \mathit{AttRepository} \rightarrow \mathit{Conf}$$

ResistanceClass *is a function $\mathit{AttCapLevel} \times \mathit{SecReqLevel} \rightarrow \mathbb{P}\,\mathit{Conf}$, mapping an attacker capabilitiy level ac_p and a security requirements level sr_q to the possible system configurations $\subseteq \mathit{Conf}$ where the security requirements are met in face of attempted attacks:*

$$\begin{aligned}
&\mathit{ResistanceClass}(ac_p, sr_q) \mapsto \\
&\quad \{\mathit{sysconf} \in \mathit{Conf} \mid \forall\, \mathit{att} \in \mathit{AttRepository} \bullet \\
&\quad\quad \mathit{NecAttCap}(\mathit{att}) \leq ac_p \Rightarrow \mathit{EvalState}(\mathit{ApplyAttack}(\mathit{sysconf}, \mathit{att})) \geq sr_q\}
\end{aligned}$$

We write $rc_{ac,sr} = ResistanceClass(ac, sr)$ *to refer to the set of system states representing the resistance class of the given attacker capability level* ac *and security requirements level* sr.

The *EvalState* and *ApplyAttack* functions are formally defined in section 6.4 in terms of the model of a generic computer system and attack repository discussed in chapters 4 and 5.

Definition 3.4.2. *A resistance class* $rc_{ac_1,sr_1} \subseteq Conf$ *is* equivalent *to a resistance class* $rc_{ac_2,sr_2} \subseteq Conf \Leftrightarrow ac_1 = ac_2 \wedge sr_1 = sr_2$. *We write* $rc_{ac_1,sr_1} = rc_{ac_2,sr_2}$.

A resistance class $rc_{ac_1,sr_1} \subseteq Conf$ *is* at least as strong *as a resistance class* $rc_{ac_2,sr_2} \subseteq Conf \Leftrightarrow ac_1 \geq ac_2 \wedge sr_1 \geq sr_2$. *We write* $rc_{ac_1,sr_1} \geq rc_{ac_2,sr_2}$.

A resistance class $rc_{ac_1,sr_1} \subseteq Conf$ *is* at most as strong *as a resistance class* $rc_{ac_2,sr_2} \subseteq Conf \Leftrightarrow ac_1 \leq ac_2 \wedge sr_1 \leq sr_2$. *We write* $rc_{ac_1,sr_1} \leq rc_{ac_2,sr_2}$.

Definition 3.4.3. *A resistance class* rc_1 *is* stronger *than a resistance class* rc_2 *if* rc_1 *is at least as strong as* rc_2 *and* rc_1 *is not equivalent to* rc_2. *We write* $rc_1 \gneqq rc_2$.

Strong resistance classes have high security requirements levels and allow for high attacker capability levels.

Lemma 3.4.1. *A security metric for security requirements* sr_m *or attacker capabilities* ac_n *is* meaningful *if* $\forall\, sr_1, sr_2 \in SecReqLevel, \forall\, ac_1, ac_2 \in AttCapLevel : rc_{ac_1,sr_1} \gneqq rc_{ac_2,sr_2} \Rightarrow rc_{ac_1,sr_1} \subsetneqq rc_{ac_2,sr_2}$.

Proof 3.4.1 (Lemma 3.4.1). *If a strengthening of security requirements or attacker capabilities does not lead to a reduction in the set of secure configurations, the two values on the security requirements or attacker capabilities scale could be merged into one without a loss of information.* □

3.5 Properties of secure software architectures

Our goal is to examine the relationship between software architectures and resistance classes as defined in the previous section. Architectural properties are studied for quantifiable items that can be used in metrics. The metrics should then help to indicate whether the program's architecture has improved from one version to the next with respect to high resistance against malware attacks.

What constitutes a *good* software architecture with respect to security has been researched earlier, e.g., in [WBG+87, Neu96, Neu00, COR04]. We group the important principles for secure software architecture into those that can be determined *locally* – for single components or connectors – or *globally*, based on an architectural description. This list is followed by attributes of minor importance and properties that are subject to interpretation and as such not easily accessible to measurement.

Following terminology outlined in [AAG95], an application's architecture consists of *components* and *connectors*. Components have *ports* to which the connectors attach. A component is of a type describing its functionality class, e.g., file or process. Connectors determine how components can be coupled, e.g., by a data transfer relationship or a code invocation relationship. Ports are the interfaces where components offer coupling. Ports used by a connector have a *role* depending on the relationship defined by the type of the connector.

Our collection of properties of secure software architectures has been mainly extracted from two sources: *Wood et al.: Computer Security* [WBG+87], and *CORAS* [COR04]. In our list we reference a source by its name, section, and item number, e.g., "Wood 6.87" refers to list item 87 in section 2.6 of [WBG+87].

3.5.1 Local security properties

The twelve local security properties discussed here can be determined at the level of a single component and its connections. A local property is independent of other components.

1. *Executable code protection* – Protection for programs/processes shall be at least as good as for the data they manage. [ACLs for executable components connected to data components shall be at least as strict as the ACLs for the connected data components.] Source: Wood 6.87, CORAS 4.25.

2. *Storage protection* – Access control is used for (intermediate) storage of data. [There exist access restrictions for components containing (intermediate) data. No direct modification of internal resources of components is possible via attached connectors from other components.] Source: Wood 8.16.

3. *Authenticity/Integrity preservation* – Authenticity/Integrity checks are used when data is imported/exported or when code is executed. [Connectors for import/export or execution are used in conjunction with authenticity roles for the connecting ports. Integrity-preserving connectors are used.] Source: Wood 2.19, 8.29, 8.30, 9.54, 9.55, CORAS 4.47, 4.48, 7.28, 7.29.

4. *Cryptographic key protection* – Access control is used for storage of cryptographic keys. [There exist access restrictions for components containing data used as a security parameter in cryptographic operations. No direct revelation of internal resources of components is possible via attached connectors from other components.] Source: CORAS 4.25.

5. *Complete access control* – All types of access are controlled, there is no access path without access control mechanisms. [All access ports of a component to which connectors could attach have access control functionality.] Source: Wood 6.4, 6.6, 6.12, 6.62, 6.91, 8.36, 9.14, 9.65.

6. *Log data protection* – Data stored in logs is protected against unauthorized modification. [Access restrictions are in place for storage components that contain data imported from logging mechanisms.] Source: Wood 9.77.

7. *Invocation logging* – Invocation of programs is logged. [All invoking connectors include capability of logging.] Source: Wood 8.35, 9.81.

8. *Sensible logging* – Security parameters, e.g., passwords, are not logged. [Logging is not enabled for connectors importing/transferring security parameters.] Source: Wood 6.17, 6.27.

9. *Process controls* – Privileged operations are only available from designated processes. [Some connectors only attach to ports of certain components. Some roles are available only for some ports.] Source: Wood 9.25.

10. *Separate input procedures* – For sensitive transactions, there exist separate input procedures that cannot be used by all processes. [There exists a trusted path for access to sensitive components, i.e., a connector requiring a human at one port, and a component having a port that accepts only a connector with the trusted path ability.] Source: Wood 7.15, CORAS 7.15.

11. *Human in the loop* – Some actions can only be initiated by persons, in contrast to processes. [Some connectors require a human at one port.] Source: Wood 2.16.

12. *Separation of privilege* – Multiple persons/Multiple processes are involved in an operation. [Multiple connectors are attached to the same port.] Source: Wood 2.15, 2.17.

3.5.2 Global security properties

The five global security properties discussed here are emergent properties of the whole system. They can only be determined if and when the complete configuration is taken into account. This affects the ability to parallelize data collection and the complexity of updating values of the metric after changes to the system.

1. *Kernelized software* – Software is structured as a single executable object that is not extended by other code objects possibly under different control. [(Main) executable component does not have invoking/importing connectors to other executable components. There might only be a single executable component.] Source: Wood 9.58.

2. *Least privilege* – As few privileges as possible are assigned to processes. [Access permissions available to a process, e.g., expressed as privileges in capability lists, are small in number.] Source: Wood 7.26, 9.79, CORAS 7.14.

3. *Least privilege sharing* – As few processes as possible are assigned a privilege. [Processes sharing an access permission, e.g., expressed as pairs in access control lists, are small in number.] Source: Wood 2.31, 6.20, 6.84, 8.56, 9.5, 9.50, 9.72, CORAS 7.10, 8.22.

4. *Consistent controls* – Security mechanisms are applied similarly and consistently. [Access permissions are applied identically for the same class of components. Component classes can be defined by origin of components, location of components, or invocation probability of components.] Source: Wood 2.26, 2.33, 6.3, 6.92, CORAS 7.8.

5. *Central software distribution* – Software is distributed centrally, i.e., from a single source. [All connectors of an exporting/transferring type, connected to executable components, have one component – probably even one port – in common.] Source: Wood 9.61.

3.5.3 Model-intrinsic security properties

Model-intrinsic security properties are local or global security properties that are not properties of a particular application, but rather properties of the environment specification or model. They are enforced regardless of applications and attacker behaviour.

1. *Controls application* – Access controls cannot be circumvented. All accesses for which access restrictions exist are checked whether or not they should be permitted. [If a component has ports with access control, then it must not be possible to attach to ports of the components that do not support access control. Connectors that support access control must not be replaceable with connectors that do not support access control.] Source: Wood 9.34, 9.72, 9.73, 9.79, CORAS 3.17, 7.13, 7.18, 7.20, 7.21, 7.22.

2. *Freshness of resources* – Residual information is not accessible to other processes. Providing access to a discarded resource ensures that the resource is put into a defined state independent from its previous content. [Creation of components or connecting to components from whose ports all connectors have been disconnected leads to the accessed component being put into an initial state without any hints to its previous content.] Source: Wood 6.83, 8.15, 9.21, 9.69.

3. *Process segregation* – Processes are segregated. They can only influence each other via their intended interface. Access of internal storage of a process is not possible from the outside. [No direct modification of a process's code and internal resources is possible by other components.] Source: Wood 6.82, CORAS 7.19.

3.5.4 Properties of minor importance

Some security properties are of lesser importance in our attack scenario (local malware attacks). Those properties are presented for completeness' sake in the following list, but not used further in this thesis.

1. *Redundant data entry* – Data from the same source is input several times using different methods. It is accepted if inputs from different methods match. [A component exposes multiple ports for the same purpose. Alternatively, multiple connectors for the same purpose are attached to a single port.] Source: Wood 7.2, 7.3, 8.10.

2. *Multifactor authentication* – Authentication is performed involving different methods. [Multiple connectors with authentication ability are attached to the same port.] Source: Wood 6.7.

3.5.5 Properties that are subject to interpretation

Not all security properties lend themselves to construction of software security metrics. In situations where security properties depend on a specific application, no generally applicable metrics can be achieved. These situations arise when the definition of the security property involves knowledge of the inner workings of a component or a connector. It is left to future work if, e.g., the introduction of *port types* could ameliorate this problem. Properties that are subject to interpretation are presented in the following list, but not used further in this thesis.

1. *User notification* – Notification or alert facilities exist to inform the user about a malfunction or security breach. [Notification components exist and are connected to the user. Connectors for relevant events are present and include monitoring abilities.] Source: Wood 9.17, CORAS 2.8, 2.13.

2. *Monitoring* – Monitoring facilities exist to detect malfunctions or security breaches. [Connectors with monitoring abilities are used for relevant events and are connected to logging or notification components.] Source: CORAS 4.41.

3. *Configuration changes logging* – Changes in logging or access control configuration are logged. [Only connectors that have logging abilities can attach to modification ports of logging configuration data or access control configuration data.] Source: Wood 6.21, 6.29, 6.84, 8.54, 9.78, CORAS 7.10.

4. *Decentralized controls* – Non-centralized control of resources is possible. [Controlling ports for access control decisions are available at decentralized components. Not all controlling connectors for resources must attach to a single controlling component.] Source: Wood 6.72, CORAS 7.19, 7.20, 7.22.

3.6 Software architecture metrics

Based on the attributes of secure and architecturally sound systems we establish a set of security metrics; these are derived from the properties of the preceding section. Each metric measures to which degree an attribute found in architecturally sound systems is satisfied. A value of 100% means that the attribute is fully satisfied, while a value of 0% means that it is not present. Changes in values between 0% and 100% can be used to track progress between versions of the same system. Our metrics can be found in the tables on the following pages. Table 3.17 provides a summary.

We present metrics for local and global security properties. Metrics for properties of minor importance are left to future work, as well as metrics for properties that require interpretation or adaptation to a specific application.

Using the metrics, our goals are twofold. Our first goal is to measure progress when comparing two versions of the same product. Metrics should help to indicate whether the program's architecture has improved from one version to the next with respect to high resistance against malware attacks. Our second goal is to examine the relation between architectural properties of a software product and the resistance classes it can be placed in. Some architectural styles might be inherently more secure than others.

Metrics have an absolute or a relative target. Those having an absolute target gauge whether or not a specific security property has been achieved and to what degree. In most cases the target will be 100 percent. Experience from security evaluation (cf., e.g., [CEM04, NIS03]) shows that metrics with absolute targets often focus on scope and rigour of security mechanisms. These metrics can also be used to find out if there exists a trend in improving security mechanisms when comparing different versions of the same application over time. Given similar functional requirements, metrics with an absolute target value may allow for comparison of the security posture of different products for the same purpose.

Relative targets are of interest where no absolute target can be set. There might be an upper or lower bound for the target value which depends on the functional requirements

of the software being evaluated. These metrics can be used to find out if there exists a trend when comparing different versions of the same application over time.

In the next sections 14 metrics derived from the principles in section 3.5 are discussed in detail in accordance with the NIST standard 800-55 [NIS03] for security metrics (see section 2.3.1 for an introduction to the standard).

3.6.1 Metrics related to security requirements

All 14 metrics in this section aim for higher values, i.e., higher values indicate better security (4 metrics have to be normalized contrary to an intuitive understanding first). Most of them count the fraction of architectural components that comply with a local architecture principle. The target for the "best" value is always 100%.

A description of each software security metric is given in separate tables 3.3 to 3.16 in NIST standard 800-55 style.

Metrics targeting data integrity: M_4*–Percentage of protected intermediate storage components* (table 3.6), M_5*–Percentage of access control instrumentation* (table 3.7), M_6*–Conformity of access permissions* (table 3.8), M_8*–Percentage of authenticity/integrity preserving connectors* (table 3.10), M_{10}*–Restriction of number of components with shared responsibility (server)* (table 3.12), M_{12}*–Percentage of trusted path connectors* (table 3.14), M_{13}*–Restriction of number of privileges* (table 3.15), M_{14}*–Restriction of number of processes sharing a privilege* (table 3.16).

Metrics targeting code integrity: M_1*–Restriction of number of executable distribution sources* (table 3.3), M_2*–Restriction of number of executable components* (table 3.4), M_3*–Percentage of protected executables* (table 3.5), M_5*–Percentage of access control instrumentation* (table 3.7), M_6*–Conformity of access permissions* (table 3.8), M_7*–Percentage of logged invocations* (table 3.9), M_8*–Percentage of authenticity/integrity preserving connectors* (table 3.10), M_{10}*–Restriction of number of components with shared responsibility (server)* (table 3.12), M_{11}*–Restriction of number of components with multiple executable extensions* (table 3.13), M_{13}*–Restriction of number of privileges* (table 3.15), M_{14}*–Restriction of number of processes sharing a privilege* (table 3.16).

Metrics targeting data confidentiality: M_4*–Percentage of protected intermediate storage components* (table 3.6), M_5*–Percentage of access control instrumentation* (table 3.7), M_6*–Conformity of access permissions* (table 3.8), M_9*–Percentage of unlogged security parameters* (table 3.11), M_{12}*–Percentage of trusted path connectors* (table 3.14), M_{13}*–Restriction of number of privileges* (table 3.15), M_{14}*–Restriction of number of processes sharing a privilege* (table 3.16).

As can be seen in table 3.18, metrics that address logging and monitoring/alert capabilities are currently not available (M_7 being the sole exception). These require interpretation of certain architectural elements, and we regard this interpretation yet as an act of evaluation. Results of the metrics would in that case be biased depending on this pre-evaluation.

3.6.2 Metrics related to attacker capabilities

All 12 metrics in this section aim for higher values, i.e., higher values indicate better security (4 metrics have to be normalized contrary to an intuitive understanding first). Most of them count the fraction of architectural components that comply with a local architecture principle. The target for the "best" value is always 100%.

Description of each software security metric is given in separate tables 3.3 to 3.16 and follows NIST standard 800-55 style.

Metrics targeting attack initiation: M_1*–Restriction of number of executable distribution sources* (table 3.3), M_2*–Restriction of number of executable components* (table 3.4), M_3*–Percentage of protected executables* (table 3.5), M_4*–Percentage of protected intermediate storage components* (table 3.6), M_5*–Percentage of access control instrumentation* (table 3.7), M_{12}*–Percentage of trusted path connectors* (table 3.14).

Metrics targeting available time: M_1*–Restriction of number of executable distribution sources* (table 3.3), M_3*–Percentage of protected executables* (table 3.5), M_4*–Percentage of protected intermediate storage components* (table 3.6), M_7*–Percentage of logged invocations* (table 3.9), M_8*–Percentage of authenticity/integrity preserving connectors* (table 3.10), M_9*–Percentage of unlogged security parameters* (table 3.11), M_{11}*–Restriction of number of components with multiple executable extensions* (table 3.13), M_{12}*–Percentage of trusted path connectors* (table 3.14).

Metrics targeting attack variation: M_1*–Restriction of number of executable distribution sources* (table 3.3), M_2*–Restriction of number of executable components* (table 3.4), M_3*–Percentage of protected executables* (table 3.5), M_4*–Percentage of protected intermediate storage components* (table 3.6), M_5*–Percentage of access control instrumentation* (table 3.7), M_8*–Percentage of authenticity/integrity preserving connectors* (table 3.10), M_9*–Percentage of unlogged security parameters* (table 3.11), M_{10}*–Restriction of number of components with shared responsibility (server)* (table 3.12), M_{11}*–Restriction of number of components with multiple executable extensions* (table 3.13), M_{12}*–Percentage of trusted path connectors* (table 3.14), M_{13}*–Restriction of number of privileges* (table 3.15), M_{14}*–Restriction of number of processes sharing a privilege* (table 3.16).

Metrics targeting influence on user: M_7*–Percentage of logged invocations* (table 3.9), M_8*–Percentage of authenticity/integrity preserving connectors* (table 3.10), M_{10}*–Restriction of number of components with shared responsibility (server)* (table 3.12), M_{12}*–Percentage of trusted path connectors* (table 3.14).

3.6.3 Discussion

All metrics and correspondence between the selected architectural properties of structurally sound and secure applications and the resulting security metrics are shown in table 3.17 and table 3.18 respectively.

Validation in chapter 6 of the metrics will focus on their use with respect to resistance classes. Architecture metrics should help to determine whether an evaluated product is close to the threshold of the next higher or lower resistance class. To that end, it will be

Table 3.3: Metric M_1: Restriction of number of executable distribution sources

Performance Goal	Protect code integrity, restrict available time to attack as well as attack initiation and attack variation capability
Performance Objective	Determine number of components from which exporting/transferring connectors originate to executable storage components
Metric	*Number of components exporting content to executable storage*
Purpose	The number of executable distribution sources gives an indication of how hard it is for an adversary to compromise an application by compromising parts of it.
Implementation Evidence	Executable components, connectors to these
Frequency	During development, after deployment, after configuration changes
Formula	$ExpConn(conn) = conn\ is\ export\ connector$ $MetricValue_0 = \#\{c \mid \exists\, c_{exe} \wedge ExpConn((c, c_{exe}))\}$ $MetricValue = \frac{1}{MetricValue_0}$
Data Source	Architectural description, source code, configuration files
Indicators	The lowest value is 1, the highest value is only bound by the complexity of the application's architecture. Lower base values are better, since they indicate a restriction in the number of attack vectors. The metric is normalized to a 0–100 percentage scale by using the reciprocal value.

of interest to have a closer look at the relationship between the software security metrics and the *attacker capabilities* and *security requirements* components of a resistance class.

Some obvious relationships exist for the following metrics:

- M_3–*Percentage of protected executables* and M_4–*Percentage of intermediate storage components* directly address code and data integrity.

- M_7–*Percentage of logged invocations* directly affects the logging value on the security requirements scales.

We expect some more relationships that lead to additional hypotheses:

- M_{13}–*Restriction of number of privileges* and M_{14}–*Restriction of number of processes sharing a privilege* have a long-standing tradition as security principles. It should be assessed if they offer useful information in our attack scenario (where a malicious process is already executed).

- M_6–*Conformity of access permissions* might be sufficient as an indicator and substitute M_2–*Limitation of number of executable components* and M_1–*Number of executable distribution sources* by reducing the raw number of components or distribution sources to the number of classes sharing identical permissions.

Coverage-type metrics can operate on the raw number of components/connectors or they can operate on weighted values for these sets. Weights could be assigned, e.g., based on module usage, module importance or module value. However, these weights tend to be highly subjective (cf. [HMC78, HHA04]) and difficult to measure.

Table 3.4: Metric M_2: Restriction of number of executable components

Performance Goal	Protect code integrity, restrict attack variation capability
Performance Objective	Determine number of code components in use by an application
Metric	*Number of connected executable components*
Purpose	The number of executable components gives an indication of how hard it is for an adversary to compromise an application by compromising parts of it.
Implementation Evidence	Executable components, connectors between these
Frequency	During development, after deployment, after configuration changes
Formula	$RelExeConn = \{(c_1, c_2) \mid (c_1, c_2)\ \textit{is execute connector}\}$ $RelExeConnTC = RelExeConn^*$ $MetricValue_0 = \#\{\operatorname{dom} RelExeConnTC \cup \operatorname{ran} RelExeConnTC\}$ $MetricValue = \frac{1}{MetricValue_0}$
Data Source	Architectural description, source code, configuration files
Indicators	The lowest value is 1, the highest value is only bound by the complexity of the application's architecture. Lower base values are better, since fewer executable components equate a lower number of possible attack vectors. The metric is normalized to a 0–100 percentage scale by using the reciprocal value.

We object to assigning weights based on importance of modules. The velocity with which attack paths are removed – and hence the velocity with which coverage climbs to 100% – is typically not assigned the same weights. Vulnerabilities in more important modules are not necessarily easier or harder to remove only because their modules are more important. In cases where the target is 100%, it is often only of interest if the value is 0%, 100%, or some value in between and whether the trend is positive or negative when comparing multiple versions. In lack of information about how long it takes to remove the remaining insecurities in a product, assigning weights to modules yields arbitrary values without significant predictive value.

Some areas are not covered by software security metrics in this thesis and are subject to future work. These include whether a layered system *per se* provides better security (cf. [Neu96, Neu00]) and whether number or percentage of applied security patterns (cf. [FP01]) have an influence on overall security of a product.

Table 3.5: Metric M_3: Percentage of protected executables

Performance Goal	Protect code integrity, restrict attack initiation and attack variation capability
Performance Objective	Determine fraction of executable components protected against modification by malware compared with all executable components of an application
Metric	$\frac{Number\ of\ protected\ executable\ components}{Number\ of\ executable\ components}$
Purpose	The fraction of executable components protected against modification gives an indication of how hard it is for an adversary to find and manipulate an executable component and, hence, threaten its integrity, i.e., its operating according to its intended specification.
Implementation Evidence	Access control lists of storage objects for executable components
Frequency	During deployment planning, after deployment, after configuration changes
Formula	$ConnTo(source) = \{target \mid (source, target) \in Connectors\}$ $ConnDataTo(source) = ConnTo(source) \cap DataComponents$ $MetricValue = \frac{\#\{c \mid c\ is\ executable \wedge \forall\ cc \in ConnDataTo(c) \bullet ACL(cc) \subseteq ACL(c)\}}{\#\{c \mid c\ is\ executable\}}$
Data Source	Architectural description (e.g., deployment diagrams) during deployment planning, file system after deployment
Indicators	The lowest value is 0, the highest value is 100%. Higher values are better, since they indicate a higher coverage and in turn a lower number of possible attack vectors.

Table 3.6: Metric M_4: Percentage of protected intermediate storage components

Performance Goal	Protect data integrity and confidentiality
Performance Objective	Determine fraction of data components protected against modification and disclosure by malware compared with all data components of an application
Metric	$\frac{Number\ of\ protected\ data\ components}{Number\ of\ data\ components}$
Purpose	The fraction of data components protected against modification and disclosure gives an indication of how hard it is for an adversary to find and manipulate a data storage component and, hence, threaten its integrity, or to retrieve sensitive information from it, i.e., threaten its confidentiality.
Implementation Evidence	Access control lists of storage objects for data components
Frequency	During deployment planning, after deployment, after configuration changes
Formula	$MetricValue = \frac{\#\{c \mid c\ is\ data\ storage \wedge ACL(c) \neq \varnothing\}}{\#\{c \mid c\ is\ data\ storage\}}$
Data Source	Architectural description (e.g., deployment diagrams) during deployment planning, file system after deployment
Indicators	The lowest value is 0, the highest value is 100%. Higher values are better, since they indicate a higher coverage and in turn a lower number of possible attack vectors.

Table 3.7: Metric M_5: Percentage of access control instrumentation

Performance Goal	Protect code integrity and data integrity and confidentiality, restrict attack variation capability
Performance Objective	Determine fraction of access ports that have access control functionality
Metric	$\frac{Number\ of\ access\ ports\ with\ access\ control}{Number\ of\ access\ ports}$
Purpose	The fraction of controlled ports gives an indication of how hard it is for an adversary to find a way around access control protection.
Implementation Evidence	Connectors between executable and data/executable components, ports of these components
Frequency	During development, after deployment, after configuration changes
Formula	$MetricValue = \frac{\#\{p \mid p\ is\ access\ port \wedge role(p) \in AccessControlRoles\}}{\#\{p \mid p\ is\ access\ port\}}$
Data Source	Architectural description, source code, configuration files
Indicators	The lowest value is 0, the highest value is 100%. Higher values are better, since they indicate a higher coverage and in turn a lower number of possible attack vectors.

Table 3.8: Metric M_6: Conformity of access permissions

Performance Goal	Protect code integrity and data integrity and confidentiality
Performance Objective	Determine highest percentage of identical access permissions set for components in the same class
Metric	$\frac{Highest\ number\ of\ identical\ access\ permissions\ for\ components\ in\ a\ class}{Number\ of\ components\ in\ a\ class}$
Purpose	The conformity of access permissions gives an indication of how hard it is for an adversary to find and manipulate a component that is worse protected than its peers.
Implementation Evidence	Access permissions, access control matrix
Frequency	During development, after deployment, after configuration changes
Formula	$CompClass \subseteq Components \hat{=} component\ class\ of\ concern$ $ACLSet = \{ACL(c) \mid c \in CompClass\}$ $max_{acl \in ACLSet}(\#\{c \mid ACL(c) = acl\})$ $MetricValue = \frac{max_{acl \in ACLSet}\ (\#\{c \mid ACL(c) = acl \wedge c \in CompClass\})}{\#\{c \mid c \in CompClass\}}$
Data Source	Architectural description, source code, configuration files
Indicators	The lowest value is 0, the highest value is 100%. Higher values are better, since they indicate a similar (high) level of protection for components of same concern and, hence, represent a lower number of possible attack vectors.

Table 3.9: Metric M_7: Percentage of logged invocations

Performance Goal	Protect data integrity and confidentiality, restrict available time to attack
Performance Objective	Determine fraction of invocations that have logging functionality
Metric	$\frac{Number\ of\ invocations\ that\ are\ logged}{Number\ of\ invocations}$
Purpose	The fraction of logged invocations gives an indication of how hard it is for an adversary to execute malicious code without detection.
Implementation Evidence	Connectors between executable and data/executable components, ports of these components
Frequency	During development, after deployment, after configuration changes
Formula	$MetricValue = \frac{\#\{c \mid c\ is\ executable\ connector \wedge has\ logging\ functionality\}}{\#\{c \mid c\ is\ executable\ connector\}}$
Data Source	Architectural description, source code
Indicators	The lowest value is 0, the highest value is 100%. Higher values are better, since they indicate a higher coverage and in turn a lower number of possible attack vectors.

Table 3.10: Metric M_8: Percentage of authenticity/integrity preserving connectors

Performance Goal	Protect data and code integrity, restrict available time to attack and attack variation capability
Performance Objective	Determine fraction of connectors used for import, export, or execution that incorporate authenticity or integrity checks prior to main operation
Metric	$\frac{Number\ of\ authenticity/integrity\ preserving\ connectors}{Number\ of\ connectors}$
Purpose	The fraction of authenticity/integrity preserving connectors gives an indication of how hard it is for an adversary to modify or substitute data or executable components without detection.
Implementation Evidence	Connectors between executable and data/executable components, ports of these components
Frequency	During development, after deployment, after configuration changes
Formula	$ImpExpExecConn(conn) = conn\ is\ import/export/execute\ connector$ $IntPres(conn) = conn\ preserves\ integrity$ $AuthPres((p_1, p_2)) = AuthRole(p_1) \wedge AuthRole(p_2)$ $MetricValue = \frac{\#\{c=(p_1,p_2) \mid ImpExpExecConn(c) \wedge (IntPres(c) \vee AuthPres((p_1,p_2)))\}}{\#\{c \mid ImpExpExecConn(c)\}}$
Data Source	Architectural description, source code, configuration files
Indicators	The lowest value is 0, the highest value is 100%. Higher values are better, since they indicate a higher coverage and in turn a lower number of possible attack vectors.

Table 3.11: Metric M_9: Percentage of unlogged security parameters

Performance Goal	Protect data confidentiality, restrict available time to attack and attack variation capability
Performance Objective	Determine fraction of transferred security parameters that are not logged
Metric	$\frac{Number\ of\ transfers\ that\ are\ not\ logged}{Number\ of\ transfers}$
Purpose	The fraction of unlogged security parameters gives an indication of how hard it is for an adversary to retrieve security parameters, e.g., passwords, from log storage.
Implementation Evidence	Connectors for data transfer from security sensitive data components
Frequency	During development, after deployment, after configuration changes
Formula	$MetricValue = \frac{\#\{c \mid c\ is\ transferring\ connector \wedge does\ not\ have\ logging\ functionality\}}{\#\{c \mid c\ is\ transferring\ connector\}}$
Data Source	Architectural description, source code, configuration files
Indicators	The lowest value is 0, the highest value is 100%. Higher values are better, since they indicate a higher coverage and in turn a lower number of possible attack vectors.

Table 3.12: Metric M_{10}: Restriction of number of components with shared responsibility (server)

Performance Goal	Protect data integrity, restrict attack variation capability
Performance Objective	Determine number of components with ports that are used in *execute* connector instances with multiple source components
Metric	$\frac{Number\ of\ executable\ components\ that\ are\ targets\ of\ two\ or\ more\ execute\ connectors}{Number\ of\ executable\ components}$
Purpose	The number of components with shared responsibility (server) gives an indication of how hard it is for an adversary to compromise an application by compromising parts of it.
Implementation Evidence	Ports, connector instances
Frequency	During development, after deployment, after configuration changes
Formula	$ClientTo(tp) = \{s \mid (s,t) \in Executable\ conn. \wedge tp \in t.ports\}$ $MetricValue_0 = \#\{c \mid \exists\, p \in c.ports \bullet \#ClientTo(p) \geq 2\}$ $MetricValue = \frac{1}{MetricValue_0 + 1}$
Data Source	Architectural description, source code, configuration files
Indicators	The lowest value is 0, the highest value is only bound by the complexity of the application's architecture and its number of executable components. Lower base values are better, since fewer targets are easier to authenticate than more targets. The metric is normalized to a 0–100 percentage scale by using the reciprocal value.

Table 3.13: Metric M_{11}: Restriction of number of components with multiple executable extensions

Performance Goal	Protect data integrity, restrict attack variation capability
Performance Objective	Determine number of components with ports that are used in connector instances with multiple target components
Metric	$\frac{\textit{Number of executable components that are sources of two or more execute connectors}}{\textit{Number of executable components}}$
Purpose	The number of components with multiple code extensions gives an indication of how hard it is for an adversary to compromise an application by compromising parts of it.
Implementation Evidence	Ports, connector instances
Frequency	During development, after deployment, after configuration changes
Formula	$SrvTo(sp) = \{t \mid (s,t) \in Executable\ conn. \land sp \in s.ports\}$ $MetricValue_0 = \#\{c \mid \exists\, p \in c.ports \bullet \#SrvTo(p) \geq 2\}$ $MetricValue = \frac{1}{MetricValue_0 + 1}$
Data Source	Architectural description, source code, configuration files
Indicators	The lowest value is 0, the highest value is only bound by the complexity of the application's architecture and its number of executable components. Lower base values are better, since fewer components executing code from multiple sources are easier to control than more. The metric is normalized to a 0–100 percentage scale by using the reciprocal value.

Table 3.14: Metric M_{12}: Percentage of trusted path connectors

Performance Goal	Protect data integrity and data confidentiality, restrict attack initiation, attack variation and user influence capability
Performance Objective	Determine fraction of import/export/execute connectors that have trusted path capability and connect the local human user with a component
Metric	$\frac{Number\ of\ trusted\ path\ connectors}{Number\ of\ connectors}$
Purpose	The fraction of trusted path connectors gives an indication of how hard it is for an adversary to eavesdrop on or manipulate communication between a sensitive component and the local human user.
Implementation Evidence	Connectors for data transfer to and from security sensitive data components
Frequency	During development, after deployment, after configuration changes
Formula	$ImpExpExecConn(conn) = conn\ is\ import/export/execute\ connector$ $TPCap(conn) = conn\ has\ trusted\ path\ capability$ $ConnUser(conn) = conn\ connects\ to\ user\ at\ one\ end$ $TPConn(conn) = TPCap(conn) \wedge ConnUser(conn)$ $MetricValue = \frac{\#\{cn \mid ImpExpExecConn(cn) \wedge TPConn(cn)\}}{\#\{cn \mid ImpExpExecConn(cn)\}}$
Data Source	Architectural description, source code, configuration files
Indicators	The lowest value is 0, the highest value is 100%. Higher values are better, since they indicate a higher coverage and in turn a lower number of possible attack vectors.

Table 3.15: Metric M_{13}: Restriction of number of privileges

Performance Goal	Protect data integrity and code integrity, restrict attack variation capability
Performance Objective	Determine average number of privileges assigned to a process
Metric	$\frac{Number\ of\ privileges\ assigned\ to\ processes}{Number\ of\ processes}$
Purpose	The average number of privileges gives an indication of how hard it is for an adversary to compromise an application once a part of it has been attacked successfully.
Implementation Evidence	Access permissions, capability lists, access control matrix
Frequency	During development, after deployment, after configuration changes
Formula	$MetricValue_0 = \frac{\sum_{processes} privileges\ assigned\ to\ process}{\#\{p \mid p\ is\ process\}}$ $MetricValue = \frac{1}{MetricValue_0 + 1}$
Data Source	Architectural description, source code, configuration files
Indicators	The lowest value is 0, the highest value is only bound by the system in which the application is executed. Lower base values are better, since they indicate a restriction in the capabilities of a process that could be a possible attack vector. The metric is normalized to a 0–100 percentage scale by using the reciprocal value.

Table 3.16: Metric M_{14}: Restriction of number of processes sharing a privilege

Performance Goal	Protect data integrity and code integrity, restrict attack variation capability
Performance Objective	Determine average number of processes assigned permissions to objects
Metric	$\frac{Number\ of\ processes\ in\ ACL\ of\ storage\ component}{Number\ of\ storage\ components}$
Purpose	The average number of processes sharing a privilege gives an indication of how hard it is for an adversary to compromise an application by compromising parts of it.
Implementation Evidence	Access control lists, access control matrix
Frequency	During development, after deployment, after configuration changes
Formula	$MetricValue_0 = \frac{\#\{(proc,c,priv) \mid (proc,priv) \in ACL(c) \wedge c\ is\ storage\ component\}}{\#\{(c,priv) \mid c\ is\ storage\ component \wedge (\bullet,priv) \in ACL(c)\}}$ $MetricValue = \frac{1}{MetricValue_0}$
Data Source	Architectural description, source code, configuration files
Indicators	The lowest value is 1, the highest value is only bound by the complexity of the application's architecture. Lower base values are better, since they indicate a restriction in the capabilities of a process that could be a possible attack vector. The metric is normalized to a 0–100 percentage scale by using the reciprocal value.

Table 3.17: Software metrics for secure architectures

Metric	Formula	Target
M_1: Centralisation of executable distribution sources	$ExpConn(conn) = conn\ is\ export\ connector$ $MetricValue = \frac{1}{\#\{c \mid \exists\, c_{exe} \wedge ExpConn((c, c_{exe}))\}}$	100%
M_2: Restriction of number of exec. comp.	$RelExeConn = \{(c_1, c_2) \mid (c_1, c_2)\ is\ execute\ connector\}$ $RelExeConnTC = RelExeConn^*$ $MetricValue = \frac{1}{\#\{\mathrm{dom}\ RelExeConnTC \cup \mathrm{ran}\ RelExeConnTC\}}$	100%
M_3: Percentage of protected executables	$ConnTo(source) = \{target \mid (source, target) \in Connectors\}$ $ConnDataTo(source) = ConnTo(source) \cap DataComponents$ $MetricValue = \frac{\#\{c \mid c\ is\ executable \wedge \forall\, cc \in ConnDataTo(c) \bullet ACL(cc) \subseteq ACL(c)\}}{\#\{c \mid c\ is\ executable\}}$	100%
M_4: Percentage of prot. intermediate storage comp.	$MetricValue = \frac{\#\{c \mid c\ is\ data\ storage \wedge ACL(c) \neq \varnothing\}}{\#\{c \mid c\ is\ data\ storage\}}$	100%
M_5: Percentage of access control instrumentation	$MetricValue = \frac{\#\{p \mid p\ is\ access\ port \wedge role(p) \in AccessControlRoles\}}{\#\{p \mid p\ is\ access\ port\}}$	100%
M_6: Conformity of access permissions	$CompClass \subseteq Components \hat{=} componentclassofconcern$ $ACLSet = \{ACL(c) \mid c \in CompClass\}$ $max_{acl \in ACLSet}(\#\{c \mid ACL(c) = acl\})$ $MetricValue = \frac{max_{acl \in ACLSet}\ (\#\{c \mid ACL(c) = acl \wedge c \in CompClass\})}{\#\{c \mid c \in CompClass\}}$	100%
M_7: Percentage of logged invocations	$MetricValue = \frac{\#\{c \mid c\ is\ executable\ connector \wedge has\ logging\ functionality\}}{\#\{c \mid c\ is\ executable\ connector\}}$	100%
M_8: Percentage of auth./integrity preserving connectors	$ImpExpExecConn(conn) = conn\ is\ import/export/exec.\ connector$ $IntPres(conn) = conn\ preserves\ integrity$ $AuthPres((p_1, p_2)) = AuthRole(p_1) \wedge AuthRole(p_2)$ $MetricValue = \frac{\#\{c = (p_1, p_2) \mid ImpExpExecConn(c) \wedge (IntPres(c) \vee AuthPres((p_1, p_2)))\}}{\#\{c \mid ImpExpExecConn(c)\}}$	100%
M_9: Perc. of unlogged sec.s parameters	$MetricValue = \frac{\#\{c \mid c\ is\ transferring\ connector \wedge without\ logging\ functionality\}}{\#\{c \mid c\ is\ transferring\ connector\}}$	100%
M_{10}: Restr. of number of comp. with shared responsibility (srv.)	$PortConnTo(sp) = \{t \mid (s, t) \in Connectors\} \wedge sp \in s.ports$ $MetricValue = \frac{1}{\#\{c \mid \exists\, p \in c.ports \bullet PortConnTo(p) \geq 2\} + 1}$	100%
M_{11}: Restr. of number of comp. with multiple exec. ext.	$SrvTo(sp) = \{t \mid (s, t) \in Executable\ conn. \wedge sp \in s.ports\}$ $MetricValue = \frac{1}{\#\{c \mid \exists\, p \in c.ports \bullet \#SrvTo(p) \geq 2\} + 1}$	100%
M_{12}: Percentage of trusted path connectors	$ImpExpExecConn(conn) = conn\ is\ import/export/exec.\ connector$ $TPCap(conn) = conn\ has\ trusted\ path\ capability$ $ConnUser(conn) = conn\ connects\ to\ user\ at\ one\ end$ $TPConn(conn) = TPCap(conn) \wedge ConnUser(conn)$ $MetricValue = \frac{\#\{cn \mid ImpExpExecConn(cn) \wedge TPConn(cn)\}}{\#\{cn \mid ImpExpExecConn(cn)\}}$	100%
M_{13}: Restriction of num. of priv.	$MetricValue = \frac{\#\{p \mid p\ is\ process\}}{\sum_{processes} privileges\ assigned\ to\ process + 1}$	100%
M_{14}: Restriction of num. of processes sharing a priv.	$MetricValue = \frac{\#\{(c, priv) \mid c\ is\ storage\ component \wedge (\bullet, priv) \in ACL(c)\}}{\#\{(proc, c, priv) \mid (proc, priv) \in ACL(c) \wedge c\ is\ storage\ component\}}$	100%

Table 3.18: Correspondence between architectural properties and security metrics

	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}	M_{11}	M_{12}	M_{13}	M_{14}
Central software distribution	M_1													
Kernelized software		M_2												
Executable code protection			M_3											
Consistent controls			M_3			M_6								
Storage protection				M_4										
Cryptographic key protection				M_4										
Log data protection				M_4										
Complete access control					M_5									
Invocation logging							M_7							
Auth./Integrity preservation								M_8						
Sensible logging									M_9					
Separation of privilege										M_{10}	M_{11}			
Separate input procedures												M_{12}		
Human in the loop												M_{12}		
Least privilege													M_{13}	
Least privilege sharing														M_{14}
Process controls	No metric (depends on functionality)													
Controls application	No metric (model-intrinsic)													
Freshness of resources	No metric (model-intrinsic)													
Process segregation	No metric (model-intrinsic)													
Redundant data entry	No metric (minor importance)													
Multifactor authentication	No metric (minor importance)													
User notification	No metric (needs interpretation)													
Monitoring	No metric (needs interpretation)													
Configuration protection	No metric (needs interpretation)													
Decentralized controls	No metric (needs interpretation)													

Chapter 4

Model of a generic computer system

Chapter summary: In this chapter we present an abstract model of a generic computer system. The model is based on the terminology used in established security criteria and research efforts. Our model consists of almost fourty types of components and connectors and captures the operating system, data, processes, and the local human user. It is the preparation for a formal specification of a generic computer system in the next chapter.

4.1 Modeling approach

We develop a model of a generic computer system based on the accumulated knowledge of three established best practices documents: the *Common Criteria for Information Technology Security Evaluation* [CC299b], Wood et al. *Computer Security* [WBG+87], and the European *CORAS* project [COR04]. Our scope is a single system, e.g., a workstation, in which benign and malicious processes co-exist. This restriction points out the contrast to large networks with many hosts – which are not our focus. The model's purpose is a description and analysis of attacks by malicious software on a program's architecture (i.e., its components and their relationships).

4.1.1 Source: Common Criteria (CC)

In a first step we derive the abstract assumptions about a computer system that the Common Criteria are based on. The CC represent the security functional requirements and terminology known and agreed to be of value at the time of release. This means that we base our model on knowledge agreed upon by many experts in the field.

The resulting model is documented in UML (figure 4-1) to get an overview. We later use the formal modeling language Z when we are concerned with a formal notation amenable to reasoning (cf. chapter 5). Apart from these described components more complex behaviour is captured in sequence diagrams and activity diagrams (that are not shown).

4.1.1.1 Point of view and structure of the CC

The approach of the Common Criteria for Information Technology Security Evaluation is to provide an internationally accepted framework for evaluation of IT security. It is divided into three parts: the first part introduces the general model [CC299a], the second concentrates on the security functional requirements [CC299b], and the third addresses

assurance requirements [CC299c]. The idea is that product acquirers specify their security needs in implementation-independent *protection profiles*. Vendors can then develop products that meet these needs. A *security target* is the implementation-specific definition of a product's security requirements. It can be subjected to evaluation by an independent body. Evaluators then check the provided documents and perform tests on the product to conclude whether the target of evaluation fulfils the desired security goals at the desired level of assurance.

The CC contain in its second part a set of constructs and language to express information security aspects. These constructs are divided into a hierarchy of *classes*, *families*, *components*, and *elements* with the intention to help locating specific security requirements.

A *class* is the most general group. All its members share a common focus, e.g., identification and authentication. Class members are called families.

A *family* is a group of sets of security requirements with shared objectives, differing in emphasis or rigour, e.g., user authentication versus user-subject binding. Family members are called components.

A *component* is a specific set of security requirements and the smallest selectable such set in the CC. Components can be ordered in a family to represent levels of strength or size of scope. Components are constructed from individual elements.

An *element* is the lowest level of expression of security requirements in the CC. In security evaluations it is verified if a target of evaluation possesses the selected elements.

The language used to express security requirements in the CC is abstract compared with specific computer systems. Hence it is possible to apply the CC to a wide range of products. For our purposes we extract the entities used in security requirements definitions (*elements*) and their relations among each other.

4.1.1.2 Relevant classes and families

With respect to attacks by malicious software (*malware*), some CC families are more relevant than others. We selected 5 classes (out of 11) and 18 families (out of 66) to be included in building our model, as shown in table 4.1. Some CC classes are not used in this iteration of the model (cf. table 4.2). They mostly cover aspects of distributed systems, cryptography, or non-technical aspects unrelated to architecture.

Model components are extracted from the specification of security functional requirements. For a couple of relations the annex to the CC has to be consulted. Some behaviour and coupling or dependencies of components is to be discerned by the software developer. In that case, all combinations are taken into account in the model.

4.1.1.3 Assessment of Common Criteria's suitability

The CC leave a great degree of freedom to a system designer. E.g., they command that there be subjects, objects, and operations, but they do not dictate specific classes of these artifacts (cf. figure 4-1). When we tried to formulate some well-known attacks from the literature [LABMC94], we encountered this underspecificity to be a hindrance. On the other hand, in our attack scenario, the malicious process and the attacked process are executed during the same session. Hence, the notion of a session is not necessary in our model.

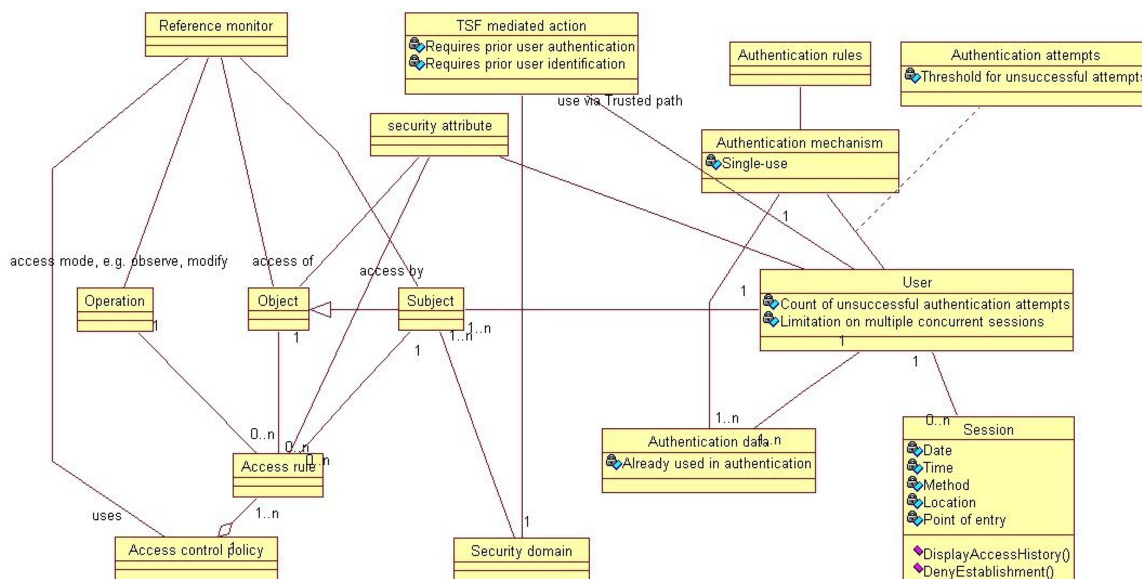


Figure 4-1: Computer system model based on selected Common Criteria security functional requirements

4.1.2 Source: Wood et al.: Computer Security

Our studies of security best practices documents, especially [WBG+87] and [COR04], suggest that adding a notion of files, memory, processes, (discretionary) access control, and user interface functions is helpful in dealing with real world examples of malware attacks (examples are presented in section 7.2). This is confirmed also by examination of, e.g., [Neu96], [YJB97], [Neu00], [Lan06b].

The checklists contained in [WBG+87] emerged from a project carried out for the United States Air Force in the 1980s. They were derived from a broad examination of the then-current literature and computer systems, and from discussions with computer security experts. Despite its age the questions raised are fundamental and still pertain to computer systems today. In total the lists consist of 738 questions in 11 categories pertaining to computer security. Out of these, 5 categories were selected for further study: *systems development*, *data and program access*, *input/output*, *processing operations*, *database and systems software*. As a result, 97 questions were evaluated as relevant as regards malware and software/system architecture (cf. appendix A).

4.1.3 Source: CORAS project

The questionnaires contained in [COR04] were used in the EU research project CORAS (*A Platform for Risk Analysis of Security Critical Systems*). With the list, an operator of the risk analysis tools is aided in addressing the security aspects of the system under evaluation. In total the lists consist of 176 questions in 8 categories pertaining to computer security on various levels (high level management down to technical aspects). Out of these, 5 categories were selected for further study: *Human*, *Physical*, *Information*, *Software*, *Exceptional circumstances*. As a result, 51 questions were evaluated as relevant as regards malware and software/system architecture (cf. appendix A).

Table 4.1: Common Criteria families used to build model

Class	*Used families*	*Unused families*
FDP *User data protection*	*Access control policy*, *Access control functions*, *Import from outside TSF control*, *Internal TOE transfer*, *Inter-TSF user data confidentiality transfer protection*	Other families in class unused because they relate to distributed systems or to implementation considerations.
FIA *Identification and authentication*	*Authentication failures*, *User attribute definition*, *User authentication*, *User identification*, *User-subject binding*	*Specification of secrets* (relates to implementation considerations, not to architecture).
FPT *Protection of the TSF*	*Internal TOE TSF data transfer*, *Domain separation*	Other families in class unused because they focus on distributed systems, implementational aspects, or do not add new objects or relations to the model.
FTA *TOE (target of evaluation) access*	*Limitation on scope of selectable attributes*, *Limitation on multiple concurrent sessions*, *Session locking*, *TOE access history*, *TOE session establishment*	*TOE access banners* (applies to human users only, not relevant for malware).
FTP *Trusted path/channels*	*Trusted path*	*Inter-TSF trusted channel* (relevant only for distributed systems).

4.2 Scope and elements of model

In general, we structure our presentation and discussion following the terminology of [AAG95]. Architecture is described by way of typed components and connectors. This abstract concept is fleshed out by defining component and connector types to describe operating system parts and operations.

Components are the basic building blocks. They perform some function and have *ports* to interact with other components. Interaction is modeled by *connectors* that attach to ports. A port in a connection acts in a certain *role*, e.g., as sender or receiver. A software or system architecture is then described as a *configuration*. Configurations consist of component instantiations (binding names and components), connector instantiations (binding names and connectors), and attachments (binding role instantiations and port instantiations).

We present the component and connector types in the next section, while we leave the formal description in Z of the more detailed model to chapter 5.

4.2.1 Scope

We are concerned with local security-sensitive programs. Even if typical access control systems associate processes with the users who created them, not all processes of the same user are equal. Software may be an intermediary to other objects or services, i.e., it brokers requests as a *Protected Subsystem*, cf. [SS75], or *Unix*-style `suid` programs.

Table 4.2: Unused Common Criteria classes to build model

Class	*Relevancy*	*Unused because*
FAU *Security audit*	Detecting malware attacks after the fact	Should be included in a later iteration
FCO *Communication*	Mostly for distributed systems as it applies to communication crossing product boundaries	Out of scope
FCS *Cryptographic support*	For implementation and distributed systems	Out of scope
FMT *Security management*	Non-technical aspects	Out of scope
FPR *Privacy*	User-related	Out of scope
FRU *Resource utilisation*	Fault tolerance and usage quotas	Out of scope

Storage controlled by a process may contain security-relevant data, e.g., credentials or copyright-restricted material received from a remote service.

4.2.2 Operating system

Our collection of software architecture building blocks used for security evaluation stems from several sources: the *Common Criteria* [CC299b], *Wood et al.: Computer Security* [WBG+87], and *CORAS* [COR04]. In this and the subsequent sections we name an item, describe its important properties and dependencies, and add the origin of each in parentheses (*CC* = [CC299b], *Wood* = [WBG+87], *CORAS* = [COR04], followed by section and item number). Checklists from *Wood* and *CORAS* are also documented in appendix A.

Software architecture building blocks related to an operating system comprise access control, integrity verification, and logging.

- 6 *Component types* (Components are the basic building blocks. They perform some function and have ports to interact with other components. With respect to entity–relationship models, components represent entities.)

  1. *Access control data* components store parameters used by a reference monitor in access control decisions. They might, e.g., represent an access control matrix or parts of it, like access control lists or capability lists. These components are typically not directly manipulated by the user and are more often read than modified. [CC, Wood 6.21, 6.29, 6.84, 8.54, 9.78, CORAS 7.10]
  2. *Logging configuration data* components store parameters that guide the logging subsystem. These components are typically not directly manipulated by the user and are more often read than modified. [CC, Wood 6.21, 6.29, 6.84, 8.54, 9.78, CORAS 7.10]
  3. 4 subject component types: *Subject–Adversary*, *Subject–Victim*, *Subject–Operating System*, and *Subject–Unspecified* are component types that are used in conjunction with the *Ownership* connector type. They represent the attacker, the studied product, the operating system, and other stakeholders, respectively. They are

needed to distinguish the studied product from its environment and to distinguish the attacker from the remainder of a system. [CC]

- 11 *Connector types* (Interaction and some attributes are modeled by connectors that attach to ports. With respect to entity–relationship models, connectors represent relationships.)

  1. *Parameter* denotes a relation between one component performing some operation and another component containing data used in a decision or calculation by the first component. Roles *Parameter processor* and *Parameter source* specify which component uses the data for a decision and which component provides the data. [CC, Wood 6.17, 6.27, CORAS 4.25]
  2. *Security parameter* is similar to *Parameter*: the data is used in a security decision or calculation. The data might, e.g., represent cryptographic keys, passwords, or tokens. [CC, Wood 6.17, 6.27, CORAS 4.25]
  3. *Integrity verification data* denotes a relation between one component containing original data and another containing data that can be used to establish whether the data contained in the first component has been altered in some way. The algorithms used could, e.g., comprise hash functions or digital signatures. Both components must be of type *Data*. Roles *Original* and *Verification data* specify which component contains the original data and which component contains the integrity verification data. [CC, Wood 2.19, 6.17, 6.27, 8.29, 8.30, 9.54, 9.55, CORAS 4.25, 4.47, 4.48, 7.28, 7.29]
  4. *Log data* denotes a relation between one component containing sensitive data or performing some operation, and another component containing log data about an access to the first component. Roles *Log data source* and *Log data target* specify which port triggers the logging and which component contains the log data. [CC, Wood 2.14, 2.16, 2.18, 2.20, 2.31, 7.25, 8.33, 8.54, 9.13, 9.14, 9.81, CORAS 4.21, 4.51, 7.10]
  5. *Monitor* denotes a relation between one component containing data or performing some operation and another component storing data or conveying data to the user or other components. Data is transferred along this connector whenever the internal state of the attached component changes based on an operation taking place at the port to which the *Monitor* connector is attached to. Roles *Monitor source* and *Monitor target* specify which component is observed for activity and which component is responsible for processing the monitoring data. [CORAS 4.41]
  6. *Ownership* denotes a relation between one component representing an owner and another representing an owned component. This associates components with an owner, i.e., *adversary*, *victim*, *operating system*, *unspecified/other*. Roles *Owner* and *Owned* specify which component owns the other. [CC]
  7. *Access–observe* denotes a relation between one component being accessible with non-modifying ("observe") access by another component. Roles *Subject* and *Object* specify which component (acting as a subject) can exercise an access right on the other (the object). [CC (access rules), Wood 6.3, 6.6, 6.64, 6.93, 9.9, 9.65, CORAS 3.19, 4.30, 7.8]

8. *Access–modify* denotes a relation between one component being accessible with modifying access by another component. Roles *Subject* and *Object* specify which component (acting as a subject) can exercise an access right on the other (the object). [CC (access rules), Wood 6.3, 6.6, 6.64, 6.93, 9.9, 9.65, CORAS 3.19, 4.30, 7.8]
9. *Access–append* denotes a relation between one component being accessible with partly-modifying ("append") access by another component. *Append* access adds data to a data component without modifying the data already present. Roles *Subject* and *Object* specify which component (acting as a subject) can exercise an access right on the other (the object). [CC (access rules), Wood 6.3, 6.6, 6.64, 6.93, 9.9, 9.65, CORAS 3.19, 4.30, 7.8]
10. *Access–delete* denotes a relation between one component being accessible with modifying ("delete") access by another component. *Delete* access removes a component completely. Roles *Subject* and *Object* specify which component (acting as a subject) can exercise an access right on the other (the object). [CC (access rules), Wood 6.3, 6.6, 6.64, 6.93, 9.9, 9.65, CORAS 3.19, 4.30, 7.8]
11. *Access–invoke* denotes a relation between one component being accessible with non-modifying ("invoke") access by another component. *Invoke* access allows the subject component to start execution of the executable data contained in the object component. Roles *Subject* and *Object* specify which component (acting as a subject) can exercise an access right on the other (the object). [CC (access rules), Wood 6.3, 6.6, 6.64, 6.93, 9.9, 9.65, CORAS 3.19, 4.30, 7.8]

- Port *types* are not necessary, only port *instances*. Ports have no inherent abstract semantics and are only needed to distinguish which roles and connectors are tied to a component and which roles work together at the same port. The function of a port is determined by the roles attached to a port involved in a connection.
  Ports can be named, though, in an architectural description. This helps to find vulnerable places in an application, i.e., when a vulnerability is detected at a port, the port's name points to the respective part of the application's architecture.

- 12 *Roles* – roles can to a great extent be taken from the connector type descriptions.

  1. *Parameter source* denotes the port of a component that provides parameter data. It is used with the *Parameter* or *Security parameter* connector. [CC, Wood 6.17, 6.27, CORAS 4.25]
  2. *Parameter processor* denotes the port of a component that processes parameter data. It is used with the *Parameter* or *Security parameter* connector. [CC (reference monitor), Wood 6.17, 6.27, CORAS 4.25]
  3. *Original* denotes the port of a component containing the original data. It is used with the *Integrity verification data* and the *Backup* connectors. [CC, Wood 2.19, 6.17, 6.27, 8.6, 8.29, 8.30, 9.54, 9.55, CORAS 4.19, 4.25, 4.47, 4.48, 7.28, 7.29]
  4. *Verification data* denotes the port of a component containg verification data for an integrity check. It is used with the *Integrity verification data* connector. [CC, Wood 2.19, 6.17, 6.27, 8.29, 8.30, 9.54, 9.55, CORAS 4.25, 4.47, 4.48, 7.28, 7.29]

5. *Log data source* denotes the port of a component that exports log data. It is used with the *Log data* connector. [CC, Wood 2.14, 2.16, 2.18, 2.20, 2.31, 7.25, 8.33, 8.54, 9.13, 9.14, 9.81, CORAS 4.21, 4.51, 7.10]
6. *Log data target* denotes the port of a component that imports or stores log data. It is used with the *Log data* connector. [CC, Wood 2.14, 2.16, 2.18, 2.20, 2.31, 7.25, 8.33, 8.54, 9.13, 9.14, 9.81, CORAS 4.21, 4.51, 7.10]
7. *Monitor source* denotes the port of a component that generates or exports monitoring data. It is used with the *Monitor* connector. [CORAS 4.41]
8. *Monitor target* denotes the port of a component that imports or presents monitoring data. It is used with the *Monitor* connector. [CORAS 4.41]
9. *Owner* denotes the port of a component that represents a stakeholder, i.e., *Owner–Adversary*, *Owner–Victim*, *Owner–Operating System*, *Owner–Unspecified*. It is used with the *Ownership* connector. [CC]
10. *Owned* denotes the port of a component that contains data or executable code. [CC]
11. *Subject* denotes the port of a component that can exercise an access right on another component. It is used with the *Access-observe*, *Access-modify*, *Access-append*, *Access-delete*, and *Access-invoke* connectors. [CC (subject), Wood 6.3, 6.6, 6.64, 6.93, 9.9, 9.65, CORAS 3.19, 4.30, 7.8]
12. *Object* denotes the port of a component on which another component can exercise an access right. It is used with the *Access-observe*, *Access-modify*, *Access-append*, *Access-delete*, and *Access-invoke* connectors. [CC (object), Wood 6.3, 6.6, 6.64, 6.93, 9.9, 9.65, CORAS 3.19, 4.30, 7.8]

4.2.3 Data

Building blocks for data cover files, folders, main memory, tapes, programs, external devices. Access control to data components is governed by operating system parts as discussed earlier.

- 3 *Component types*

  1. *Data* components store arbitrary application content processed internally or presented to the user. They might, e.g., represent files, memory, or user interface devices. [CC (object), Wood, CORAS]
  2. *Tamper-proof storage* components store arbitrary data in an external and/or tamper-proof module. They might, e.g., represent smart cards with security memory or WORM (write once read multiple) media. [Wood 6.29, 8.54]
     Tamper-proof storage components may also be substituted by data components to which only *observe* and *append* access rights exist and where the access control configuration relating to these components cannot be changed.
  3. *Reference* components point to *Data*, *Container*, or *Reference* components to which they are connected by *Reference rule* connectors. If a port of a *Reference* component is instantiated with a *Data target*, *Data source*, or *Executed* role, this role is forwarded to appropriate ports of components pointed to by the *Reference rule*. [Wood 8.8, 9.76]

- 6 *Connector types*

  1. *Data transfer* denotes a relation between one component providing data and another component receiving data. The data transferred can be arbitrary data. It can also be used to represent method invocation. To do this, command data is transferred to the component's target port responsible for method invocation. It does not matter whether data stored in a component is modified or if a method of that component is invoked, as both cases lead to a change in the internal state of the component. Roles *Data source* and *Data target* specify which component exports the data and which component imports it. Roles *Authenticated data source* and *Authenticated data target* require that the identity of the component be verified by the connector so that the perceived source/target matches the real one. [CC, Wood 2.19, 6.2, 8.29, 8.30, 9.54, 9.55, 9.61, CORAS 3.18, 3.20, 4.23, 4.45, 4.47, 4.48, 7.16, 7.17, 7.26, 7.28, 7.29]
  2. *Backup* denotes a relation between one component containing original data and another component containing a copy of said data, created at some point in time. Both components must be of type *Data*. Roles *Original* and *Copy* specify which component contains the original data and which component contains the backup data. [CORAS 4.19, Wood 8.6]
  3. *Contained–by* denotes a relation between one component containing another component. It is, e.g., used for folders or drives. Roles *Container* and *Contained* tell which component is the container and which one is contained by the container. [CC]
  4. *Reference rule–Static* denotes a relation between a *Reference* component and another component. The referencing component always points to the same referenced component. Roles *Reference source* and *Reference target* are used in conjunction with this connector type. [Wood 8.8, 9.76]
  5. *Reference rule–Search order* denotes a relation between a *Reference* component and another component. The referencing component points to the referenced component unless there exists another component with the same name as the referenced component that is contained in a *Container* component that precedes the *Container* component of the referenced component in the search order. Roles *Reference source* and *Reference target* are used in conjunction with this connector type. [Wood 8.8, 9.76]
  6. *Reference rule–Container* denotes a relation between a *Reference* component and a *Container* component. The referencing component points to all components contained in the *Container* component. Roles *Reference source* and *Reference target* are used in conjunction with this connector type. [Wood 8.8, 9.76]

- 9 additional *Roles*

  1. *Original* denotes the port of a component containing the original data (see page 77).
  2. *Copy* denotes the port of a component containg a copy of the data of another component for backup purposes. It is used with the *Backup* connector. [CORAS 4.19, Wood 8.6]

3. *Data source* denotes the port of a component that exports data. It is used with the *Data transfer* connector. [CC, Wood 2.19, 6.2, 8.29, 8.30, 9.54, 9.55, 9.61, CORAS 3.18, 3.20, 4.23, 4.45, 4.47, 4.48, 7.16, 7.17, 7.26, 7.28, 7.29]
4. *Authenticated data source* denotes the port of a component that exports data. The identity of the component must be verified by the connector. It is used with the *Data transfer* connector. [CC (authentication data), Wood 2.19, 6.2, 8.29, 8.30, 9.54, 9.55, 9.61, CORAS 3.18, 3.20, 4.23, 4.45, 4.47, 4.48, 7.16, 7.17, 7.26, 7.28, 7.29]
5. *Data target* denotes the port of a component that imports data or stands for a method invocation interface. It is used with the *Data transfer* connector. [CC, Wood 2.19, 6.2, 8.29, 8.30, 9.54, 9.55, 9.61, CORAS 3.18, 3.20, 4.23, 4.45, 4.47, 4.48, 7.16, 7.17, 7.26, 7.28, 7.29]
6. *Authenticated data target* denotes the port of a component that imports data or stands for a method invocation interface. The identity of the component must be verified by the connector. It is used with the *Data transfer* connector. [CC, Wood 2.19, 6.2, 8.29, 8.30, 9.54, 9.55, 9.61, CORAS 3.18, 3.20, 4.23, 4.45, 4.47, 4.48, 7.16, 7.17, 7.26, 7.28, 7.29]
7. *Container* denotes the port of a component acting as a container for other components. It is used with the *Contained–by* connector. [CC]
8. *Contained* denotes the port of a component that is contained by another component, the container. It is used with the *Contained–by* connector. [CC]
9. *Reference source* denotes the port of a *Reference* component where a reference rule connector is attached. [Wood 8.8, 9.76]
10. *Reference target* denotes the port of a component where a reference rule connector is attached. [Wood 8.8, 9.76]

4.2.4 Processes and IPC

Processes are executed programs, hence processes are modeled by the data components containing the executable code. They communicate with each other and the operating system via inter-process communication (IPC) mechanisms. Architectural building blocks are:

- 1 additional *Component type*

  1. *Data* components store arbitrary application content processed internally or presented to the user. They might, e.g., represent an executable file (see page 78).
  2. *Tamper-proof storage* components store arbitrary data in an external and/or tamper-proof module (see page 78).
  3. *Firmware* components store executable data in a tamper-proof module. They might, e.g., represent smart cards or EPROM, and be used to store sensitive programs. [Wood 9.53]
     Firmware components may also be substituted by data components to which only *observe* access rights exist, where the access control configuration relating to these components cannot be changed, and that are marked as executable components.

- 3 *Connector types*

  1. *Execute* denotes a relation between one component executing another. Both components must be of type *Data* (one can be of type *User*, though). Roles *Executor* and *Executed* specify which component initiates execution of the other. [CC, Wood 8.35, 9.81]
  2. *Linked* denotes a relation between one (executable) component linking another (executable). Both components must be of type *Data*. Role *Link executor* for the executing component and roles *Link executed static* and *Link executed dynamic* specify which component initiates execution of the other and whether the linked component is linked statically or dynamically. [Wood 8.8, 9.76]
  3. *Subject binding* denotes a relation between an executable component and a *subject*, e.g., as is the case for Unix `suid` programs. Access rights associated with the subject are then granted to the executable component. Roles *Subject* and *Executed* specify which component is the subject and which is the associated executable component. [CC]

- 5 additional *Roles*

  1. *Data source* denotes the port of a component that exports data (see page 80).
  2. *Authenticated data source* denotes the port of a component that exports data (see page 80).
  3. *Data target* denotes the port of a component that imports data or stands for a method invocation interface (see page 80).
  4. *Authenticated data target* denotes the port of a component that imports data or stands for a method invocation interface (see page 80).
  5. *Executor* denotes the port of a component executing another component containg executable data. It is used with the *Execute* connector. [Wood 8.35, 9.81]
  6. *Executed* denotes the port of a component containg executable data and being executed by another component. It is used with the *Execute* connector. [Wood 8.35, 9.81]
  7. *Link executor* denotes the port of a component linking and executing another component containg executable data. It is used with the *Linked* connector. [Wood 8.8, 9.76]
  8. *Link executed static* denotes the port of a component containg executable data and being statically linked and executed by another component. It is used with the *Linked* connector. [Wood 8.8, 9.76]
  9. *Link executed dynamic* denotes the port of a component containg executable data and being dynamically linked and executed by another component. It is used with the *Execute* connector. [Wood 8.8, 9.76]

4.2.5 User

The user of the local machine interacts with processes and the operating system via the user interface. While the interface can be modeled by special data components, we need another component type that stands for the user.

- 3 additional *Component types*

  1. *Local human user* components represent the user of a program. Local human user components are typically connected to *UI Output* components responsible for conveying information to and from the user. [CC (user), Wood 2.16]
  2. *UI Output* components are user interface (UI) components that display data which can be captured by the local human user. [CC (user), Wood 2.16]
     UI Output components may also be substituted by data components to which the local human user has *observe* access rights and that transfer data to the user.
  3. *UI Input* components are user interface (UI) components that receive data provided by the local human user. [CC (user), Wood 2.16]
     UI Input components may also be substituted by data components to which the local human user has *append* access rights and that transfer data from the user.
  4. *Data* components store arbitrary produced by or presented to the user (see page 78).

Some of the data presented to the user can be categorized into *security parameters*, *log data*, or data *monitored* by the user. The user can also execute programs before an attack begins.

- 3 *Connector types*

  1. *Security parameter* specifies that data is used in a security decision or calculation. It might influence the user in making a security-relevant decision (see page 76).
  2. *Log data* specifies that the data presented to the user is data about the performance and completion of an operation. It might influence the user in making a security-relevant decision or it might aid the user in detecting an attack (see page 76).
  3. *Monitor* expresses that the user might be informed about some operations in real time. It might influence the user in making a security-relevant decision or it might aid the user in detecting an ongoing attack (see page 76).
  4. *Execute* denotes a relation between one component executing another (see page 81).

4.3 Architectural description

A configuration consists of component instantiations (binding names and components), connector instantiations (binding names and connectors), and attachments (binding role instantiations and port instantiations).

Architectures in the CC Part of a CC evaluation is the analysis of the architectural design of a target of evaluation. This can be done either at high level (major structural units, i.e., subsystems) or low level (including the internal workings). The criteria request

only that the presentation of the design should be informal, semi-formal, or formal, and the level of detail that has to be documented. They do not provide an explicit model to use.

This lack of an explicit model in the CC is also pointed out by [Whi01], stating that "[t]o develop an extensible method for designing secure solutions, additional work is required to develop: 1. A system model that is representative of the functional aspects of security within complex solutions. 2. A systematic approach for creating security architectures based on the Common Criteria requirements taxonomy and the corresponding security system model."

There are two computer system models contained in the CC: monolithic and distributed products.

- A *monolithic* product (cf. [CC299b] figure 1.1) contains all security attributes and resources inside the scope of control of its security functions. Human users and remote information technology products are outside and can access the product only via the security functions' interface.
- A *distributed* product (cf. [CC299b] figure 1.2) transfers data to and from untrusted information technology products, and uses remote functions of remote products outside the security functions' interface. It consists of multiple separated parts, all connected through internal communication channels.

Other architectures are not considered explicitly in the CC documentation. However, they are implicitly supported by the security functional requirements.

Our attack scenario is probably closer to the *monolithic product* architecture. Differences are that some resources may lie outside the scope of control of the product under attack.

4.3.1 Configured and installed product

An architectural description of a configured and installed product consists of a collection of data storage and executable data components, connectors for data transfer and execution dependencies, and an access control configuration. The access control configuration contains the subjects and the *access* connectors to the components of the product.

4.3.2 Attacker

An attacker consists of a malicious program – a process while being executed – and probably some data components. The process might exchange data with a user interface, and is bound to a subject identity from which access permissions to other components can be derived.

The target components of an attacker are typically one or several data components or an executable component, in accordance with the security requirements discussed in section 3.1. An attacker aims to violate these security requirements.

4.3.3 Malware attack on a process

In addition to the architectural description of the configured and installed product, and the attacker, we need a specification of the involved operating system components. Attacks

as listed in section 3.3 and formally defined in section 6.1 define the rules by which the state of the system is attempted to be modified. I.e., an attack description details how components and connectors can be added to a system and which connectors are employed by the attacker.

4.4 Example of an architectural description

To show our model in action, we give a sample architectural description in our syntax. As an example, we use the homebanking application *StarMoney 5.0* by *Star Finanz – Software Entwicklung und Vertriebs GmbH.* We expose the software structure based on analysis without knowledge of the source code or internal design documentation. The full analysis is documented in section 7.2.

4.4.1 Components

Homebanking transactions are prepared at a local machine where they are signed by help of a smart card and then transmitted to the bank's server. Architecturally, the homebanking application with a total size of ca. `50 MB` consists of three small executable modules, `StartStarMoney.exe [60 KB]` for preparing the start of the main executable `StarMoney.exe [172 KB]`, and `SCRSetup.exe [132 KB]` for configuring smart card readers attached to the system. (five additional executable modules perform peripheral functionality: Conversion of older versions' data (`smkonv.exe`), T-Online Classic access (`CLGate32.exe`, `sfkclgateslave.exe`, `sfktonac.exe`), remote support (`NetViewer.exe`))

The executable modules are supported by 89 executable library files that extend the functionality of the main executable along 6 function groups: *Core banking*, *Smart card communication*, *Database access*, *Communication*, *User interface*, *Miscellaneous*.

4.4.2 Configuration: interaction and dependencies

We present the architectural description of the `SCRSetup.exe` utility software as a sample. This includes executable and data components, execute and data transfer connectors, and parts of the access control configuration.

In addition to the single executable file `SCRSetup.exe`, more than a dozen files are linked at runtime. These are 14 files provided by the manufacturer that are stored in the same folder as the main application, 2 third party files (`ct32.dll` and `np*.dll`), as well as 16 files that belong to the operating system (two of which are shown in the figures: `winscard.dll` and `scarddlg.dll`).

Some configuration data influences the execution of the application. It is stored in three files and imported by several modules.

Figure 4-2 shows *executable data* components and *executable* connectors of application `SCRSetup.exe`, i.e., its call graph. In the upper right-hand part of a component rectangle the container in which the component resides is shown. An arrowhead represents a *LinkExecuted* role, the tail of an arrow stands for a *LinkExecutor* role. Operating system data components that are linked are not shown in the figure. They do not offer an additional attack surface and are stored in operating system locations where they are protected by access rights against modification by malicious software with standard user rights. The only exceptions here are `scarddlg.dll` that interfaces with the local

Figure 4-2: Executable data components and execute connectors of `SCRSetup.exe`

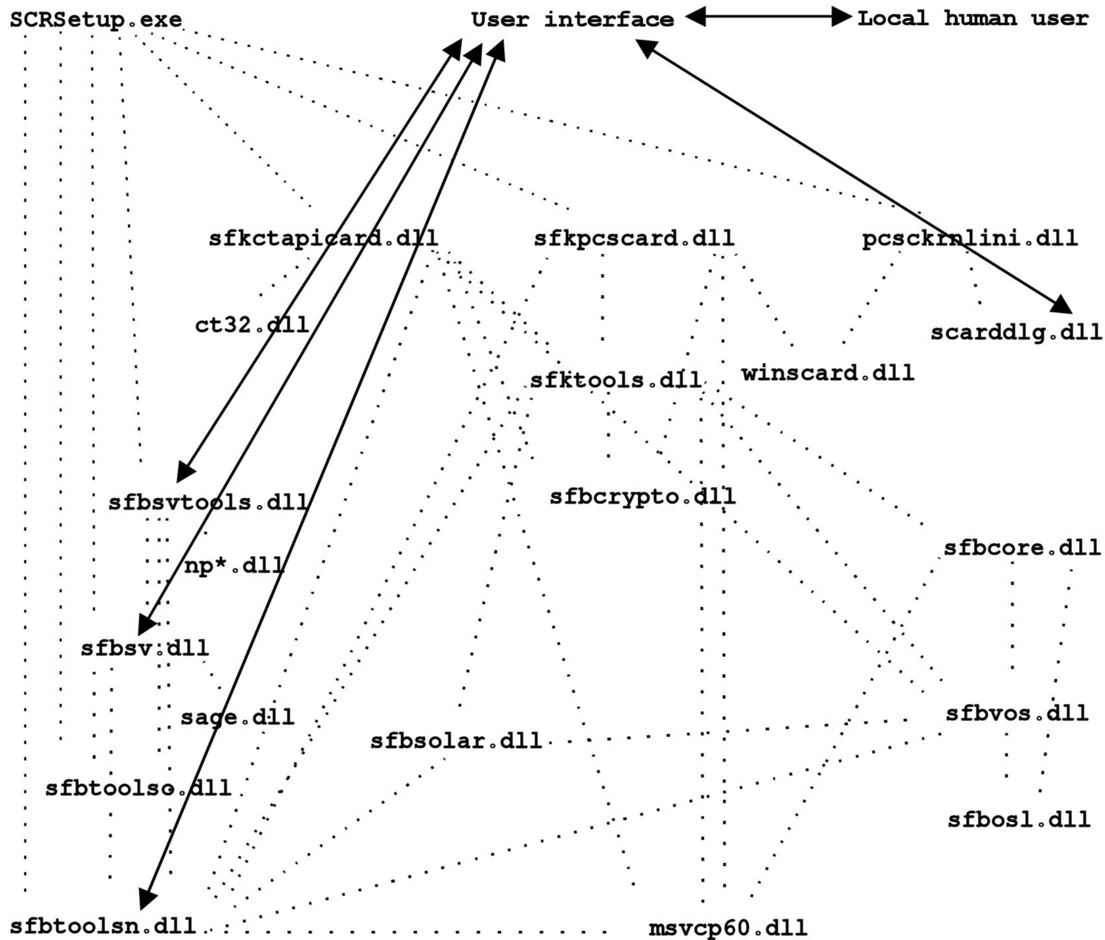


Figure 4-3: Executable data components of SCRSetup.exe interfacing with the local human user

human user and winscard.dll that could be substituted when linked with an incomplete reference. Strictly speaking, the arrows in the figure are an abbreviation for a reference connector with a reference component as target pointing a *Reference rule–Search order* to an executable data component. In addition, a filled circle represents a *Data target* role, a simple circle stands for a *Data source* role. There are no special access permissions set for these data components.

In figure 4-3 we present which components of the application communicate with the local human user via a shared user interface. Since the shared user interface does not offer protection against interference by other processes, each of the components can possibly receive malformed or forged input and must implement validation routines for the input it receives. An arrowhead represents a *data target* role. Dotted lines represent *executable* connectors, i.e., the call graph from figure 4-2.

The complete access control configuration is not shown for this sample. All components of the application enjoy the same level of protection, being either *Access-observe* $\wedge$ *Access-invoke* or *Access-modify* $\wedge$ *Access-observe* $\wedge$ *Access-invoke* for standard users, depending on the location selected during install in the Microsoft Windows operating system file system.

A comprehensive architectural description of *StarMoney* can be found in section 7.2.

Chapter 5

Formal model of generic computer system

Chapter summary: In this chapter we present our model of a generic computer system and malware attacks in the formal specification notation Z. We develop our formal model based on the model derived from the Common Criteria security functional requirements in chapter 4. Its scope is a single system, e.g., a workstation in which benign and malicious applications co-exist. This restriction points out the contrast to large networks with many hosts – which are not the focus of this work. The formal model contains all the structures needed to capture a program's architecture (i.e., its components and their relationships) and a malicious process attempting to violate security requirements of the program.

We start with a justification for the use of Z, followed by a bottom-up presentation of our formal model. We adhere to established Z style, i.e., we start the specification with given sets and global constants, followed by a definition of state, initial state, and finally operations. The last section discusses limitations of the model.

5.1 Formal specification in Z

The formal specification notation Z is based on Zermelo-Fraenkel set theory and first order predicate logic (cf. e.g. [Spi92], [Jac97], [PST96], [BSC94], [Bow96]). It has been developed by the Programming Research Group (PRG) at the Oxford University Computing Laboratory (OUCL) and elsewhere since the late 1970s. It is now defined by an ISO standard. [Z00]

More on Z and its applications to formal (security) and software architecture modeling can be found in section 2.5.3. We model a program's architecture according to [AAG95]. An architecture is modeled as components and connectors. The structure of our model presentation follows the established Z style of starting with given sets and global constants, then defining state, initial state, and finally operations.

The reader is advised that there is an index of all types and schema definitions on page 127. It can be used when browsing the specification to easily retrieve the location of used types and schemas.

5.2 Model definition

Our model of a generic computer and operating system – as presented in section 4.2 – consists of seven parts: a file system, a memory subsystem, user interface functions, a process subsystem, access control and monitoring facilities, inter-process communication capability, and the user interface.

5.2.1 Given sets and global constants

An application's architecture consists of *components* and *connectors*. Components have *ports* to which the connectors attach. A component is of a type describing its functionality class, e.g., a data storage component or a subject component. Connectors determine how components can be coupled, e.g., by a data transfer relationship or a code invocation relationship. Ports are the interfaces where components offer coupling. Ports used by a connector have a *role* depending on the relationship defined by the type of the connector. Component, connector, and role types of the architectural state model are explained in section 4.2.2.

$$
\begin{aligned}
COMPTYPE ::= &\\
& CTACONFDATA \mid CTLCONFDATA \mid \\
& CTSUBJECTADVERSARY \mid CTSUBJECTVICTIM \mid \\
& CTSUBJECTOS \mid CTSUBJECTUNSPECIFIED \mid \\
& CTDATA \mid CTTAMPERPROOFSTORAGE \mid CTFIRMWARE \mid \\
& CTREFERENCE \mid \\
& CTHUMANUSER \mid CTUIOUTPUT \mid CTUIINPUT
\end{aligned}
$$

MappableComponentTypes is the set of component types that comprise the range of the mapping functions that relate our model to an architectural description.

$$
\begin{array}{|l}
MappableComponentTypes : \mathbb{P}\ COMPTYPE \\
\hline
MappableComponentTypes = \{ \\
\quad CTSUBJECTADVERSARY, CTSUBJECTVICTIM, CTSUBJECTOS, \\
\quad CTSUBJECTUNSPECIFIED, CTDATA, CTUIOUTPUT, CTUIINPUT\}
\end{array}
$$

$$
\begin{aligned}
CONNTYPE ::= &\\
& CNDATATRANSFER \mid \\
& CNACCESSOBSERVE \mid CNACCESSMODIFY \mid \\
& CNACCESSAPPEND \mid CNACCESSDELETE \mid \\
& CNACCESSINVOKE \mid \\
& CNLOGDATA \mid CNMONITOR \mid CNSECPARAM \mid \\
& CNINTEGRITYVERIFICATIONDATA \mid \\
& CNBACKUP \mid CNCONTAINEDBY \mid \\
& CNREFERENCERULESTATIC \mid \\
& CNREFERENCERULESEARCHORDER \mid \\
& CNREFERENCERULECONTAINER \mid \\
& CNPARAM \mid \\
& CNEXECUTE \mid CNLINKEDEXEC \mid \\
& CNSUBJECTBINDING \mid CNOWNER
\end{aligned}
$$

$[PORT]$

$ROLE ::=$
$RODATASOURCE \mid ROAUTHENTICATEDDATASOURCE \mid$
$RODATATARGET \mid ROAUTHENTICATEDDATATARGET \mid$
$ROSUBJECT \mid ROOBJECT \mid$
$ROLOGDATASOURCE \mid ROLOGDATATARGET \mid$
$ROMONITORSOURCE \mid ROMONITORTARGET \mid$
$ROPARAMSOURCE \mid ROPARAMPROCESSOR \mid$
$ROORIGINAL \mid ROVERIFICATIONDATA \mid ROCOPY \mid$
$ROCONTAINER \mid ROCONTAINED \mid$
$ROREFERENCESOURCE \mid ROREFERENCETARGET \mid$
$ROEXECUTOR \mid ROEXECUTED \mid$
$ROLINKEXECUTOR \mid$
$ROLINKEXECUTEDSTATIC \mid$
$ROLINKEXECUTEDDYNAMIC \mid$
$ROOWNER \mid ROOWNED$

Objects like files etc. can be named. Names are taken from the given set *NAME* which includes the names of the component instances of the architecture model. Connector instances are also named.

$[NAME, CONNNAME]$
$COMPNAME : \mathbb{P}\, NAME$

An architectural description comprises instantiations of the component and connector types with their associated ports and roles (for terminology refer to [AAG95]). Components and connectors are named to distinguish instantiations of the same type.

$PORTINST == COMPNAME \times PORT$
$ROLEINST == CONNNAME \times ROLE$

Besides its functional requirements, an information system has non-functional security requirements. With respect to security, the ultimate goal of the system is the protection of data against unauthorized access by malicious processes. The internal structure of the data does not matter. We hence define *DATA* as a given set here. The constant *CONTENT_EMPTY* denotes a data item without content, whereas *CONTENT_EXECUTABLE* denotes data that could be executed, e.g., a program. Data or code under control of the adversary is marked as being of *attacker's choice*. When a variable assumes a value of the set *CONTENT_ATTACKERS_CHOICE*, it represents a situation

where the adversary is able to discern the variable's value.

$$
\begin{array}{l}
[DATA] \\
CONTENT_EMPTY : \mathbb{P}\, DATA \\
CONTENT_EXECUTABLE : \mathbb{P}\, DATA \\
CONTENT_ATTACKERS_CHOICE : \mathbb{P}\, DATA
\end{array}
$$

Since we are not interested in the internal structure of the data, we need a way to compare two data items with each other. The relation *appended* associates data items with those that result from appending data to the first.

$$appended : DATA \leftrightarrow DATA$$

Communication via IPC (inter-process communication) mechanisms is modeled as a function mapping input data to output data. IPC-enabled components are assigned a function of the *IPCResponse* type. A change in behaviour is then represented by a change in the assigned function. The function becomes part of the IPC-enabled component's state.

$$IPCResponse == DATA \rightarrow DATA$$

The access control subsystem regulates access of *subjects* to *objects*. Subjects are special objects, typically associated with active entities like users or processes. Four subjects are special: $SUBJECT_ADVERSARY$ represents the adversary, $SUBJECT_VICTIM$ represents the process under attack, $SUBJECT_OS$ represents the operating system, and $SUBJECT_UNSPECIFIED$ is used for subjects where it does not matter whom they represent.

$$
\begin{array}{l}
[OBJECT] \\
SUBJECT : \mathbb{P}\, OBJECT \\
SUBJECT_ADVERSARY, \\
SUBJECT_VICTIM, \\
SUBJECT_OS, \\
SUBJECT_UNSPECIFIED : SUBJECT
\end{array}
$$

We introduce *subjectComponentTypes* and *subjectComponentTypeToSUBJECT* to map subjects in the model to *subject type* components in an architectural description.

$$
\begin{array}{|l}
subjectComponentTypes : \mathbb{P}\, COMPTYPE \\
\hline
subjectComponentTypes = \{CTSUBJECTADVERSARY, CTSUBJECTVICTIM, \\
\quad CTSUBJECTOS, CTSUBJECTUNSPECIFIED\}
\end{array}
$$

$$
\begin{array}{|l}
subjectComponentTypeToSUBJECT : subjectComponentTypes \rightarrow SUBJECT \\
\hline
subjectComponentTypeToSUBJECT = \{ \\
\quad (CTSUBJECTADVERSARY, SUBJECT_ADVERSARY), \\
\quad (CTSUBJECTVICTIM, SUBJECT_VICTIM), \\
\quad (CTSUBJECTOS, SUBJECT_OS), \\
\quad (CTSUBJECTUNSPECIFIED, SUBJECT_UNSPECIFIED)\}
\end{array}
$$

Supported types of access are *creation* of objects, *observing* the content of an object, *modifying* an object, an object by *appending* data to its content, *deleting* an object, *invoking* an (executable) object (or a method thereof).

$$\begin{aligned} ACCESSMODE ::= & \\ & ACCESS_CREATE \mid ACCESS_OBSERVE \mid \\ & ACCESS_MODIFY \mid ACCESS_APPEND \mid \\ & ACCESS_DELETE \mid ACCESS_INVOKE \end{aligned}$$

$ACCESS_GENERIC_ALL$ denotes the set of all access modes. Its purpose is to facilitate access checks by having to test only for set membership.

$$ACCESS_GENERIC_ALL : \mathbb{P}\, ACCESSMODE$$

$$\begin{aligned} & ACCESS_GENERIC_ALL = \{ \\ & \quad ACCESS_CREATE, ACCESS_OBSERVE, ACCESS_MODIFY, \\ & \quad ACCESS_APPEND, ACCESS_DELETE, ACCESS_INVOKE\} \end{aligned}$$

We introduce *accessConnectorTypes* and *accessConnectorTypeToACCESSMODE* to map access modes in the model to *access type* connectors in an architectural description.

$$accessConnectorTypes : \mathbb{P}\, CONNTYPE$$

$$\begin{aligned} & accessConnectorTypes = \{ \\ & \quad CNACCESSOBSERVE, CNACCESSMODIFY, CNACCESSAPPEND, \\ & \quad CNACCESSDELETE, CNACCESSINVOKE\} \end{aligned}$$

$$accessConnectorTypeToACCESSMODE : accessConnectorTypes \rightarrow ACCESSMODE$$

$$\begin{aligned} accessConnectorTypeToACCESSMODE = \{ & \\ & (CNACCESSOBSERVE, ACCESS_OBSERVE), \\ & (CNACCESSMODIFY, ACCESS_MODIFY), \\ & (CNACCESSAPPEND, ACCESS_APPEND), \\ & (CNACCESSDELETE, ACCESS_DELETE), \\ & (CNACCESSINVOKE, ACCESS_INVOKE)\} \end{aligned}$$

An access control matrix specifies whether a specific accessmode is allowed for a certain subject/object combination. The access control policy is later specified based on an access control matrix.

$$ACM == (SUBJECT \times OBJECT) \rightarrow \mathbb{P}\, ACCESSMODE$$

Operations during a system's lifetime are recorded in a system history. This is a log of which subject components and target components were involved in a system event. A system history consists of a sequence of system events. Supported events are *creation* of components, *observation* of a component's content, *modifying* components, *renaming* components, *appending* data to a component's content, *deleting* components, and *invoking*

a component (or a method thereof).

$$[EVENTACTION]$$
$$SYSTEMEVENT == \\ \quad COMPTYPE \times COMPNAME \times EVENTACTION$$

$$EVENT_ACCESS_CREATE, \\ \quad EVENT_ACCESS_OBSERVE, \\ \quad EVENT_ACCESS_MODIFY, \\ \quad EVENT_ACCESS_RENAME, \\ \quad EVENT_ACCESS_APPEND, \\ \quad EVENT_ACCESS_DELETE, \\ \quad EVENT_ACCESS_INVOKE : EVENTACTION$$

The file system is organised as nodes, representing folders and files. A path name is a sequence of file and folder names to uniquely identify a file or folder in a tree structure.

$$[FSNODE]$$
$$PATHNAME == \operatorname{seq} NAME$$

Some objects have unique identifiers that are called *handles*. This applies inter alia to processes and memory areas.

$$[HANDLE]$$

5.2.2 State

The system state comprises the file system's state, the state of the computer's main memory, the user interface, processes and their extensions, and the access control configuration.

5.2.2.1 File system

Files are components that can be used to store data. *Folders* are components that contain files and other folders – subfolders. Files and folders form a tree structure, the *file system*.

We first show a general tree defined by the following five schemas, starting with a directed graph as described in [BSC94].

$digraph[X]$ is the set of directed graphs over X. A directed graph is a set of nodes and a relation describing the directed edges connecting some of these nodes.

$$digraph[X] == \{n : \mathbb{P}\, X;\ e : X \leftrightarrow X \mid (\operatorname{dom} e \cup \operatorname{ran} e) \subseteq n\}$$

$dag[X]$ is the set of directed acyclic graphs over X. These are digraphs that contain no cycles, i.e., the transitive closure of the edge relation does not include any vertices that are mapped to themselves.

$$dag[X] == \{n : \mathbb{P}\, X;\ e : X \leftrightarrow X \mid (n, e) \in digraph[X] \land \operatorname{disjoint}\langle e^{+}, \operatorname{id} X\rangle\}$$

$condag[X]$ is the set of connected directed acyclic graphs. Every node is joined to

every other node, in one direction or the other.

$$condag[X] == \{n : \mathbb{P}\,X;\ e : X \leftrightarrow X \mid (n, e) \in dag[X] \land (e \cup e^{\sim})^{*} = n \times n\}$$

tree$[X]$ is the set of trees. These are connected *dags* where each element is related to at most one other, its *parent*.

$$tree[X] == \{n : \mathbb{P}\,X;\ e : X \leftrightarrow X \mid (n, e) \in condag[X] \land e \in X \rightarrowtail X\}$$

The generic schema *Tree*$[N]$ is applicable for any node type N. The desired type of node is supplied when the schema is used.

The tree t consists of a non-empty set *node* of nodes, and a directed edge function, *parent*. Those nodes not in the range of *parent* are the *leaf* nodes; they have no descendants. The *root* node is the single node that has no parent. For reasons of convenience the inverse to the *parent* function is defined: the *child* relation.

Tree $[N]$

$t : tree[N]$
$parent : N \rightarrowtail N$
$node, leaf : \mathbb{P}_1\, N$
$child : N \leftrightarrow N$
$root : N$

$(node, parent) = t$
$leaf = node \setminus \operatorname{ran} parent$
$\{root\} = node \setminus \operatorname{dom} parent$
$child = parent^{\sim}$

This concludes the tree definition taken from [BSC94]. We are now going to use a tree to build up a file system, i.e., a tree with folders as the inner nodes and files as the leaf nodes.

Files are components that have a file name for reference purposes and store content (of the given set *DATA*). Files are stored in the file system at a location identified by *FSNODE*.

File

$filename : NAME$
$content : DATA$
$location : FSNODE$

Folders are components that have a folder name for reference purposes and contain files (of schema *File*) and folders (of schema *Folder*). Files are stored directly with their folder. Subfolders are not stored in the schema *Folder*; they are accessible via the function *Subfolder* of the *FileSystem* schema that retrieves subfolders of a given folder. Folders

are stored in the file system at a location identified by *FSNODE*.

All file names of files in the same folder must be unique. The same holds for subfolders; all folder names of folders in the same parent folder must be unique. Subfolders and files in the same parent folder may have the same name, though. The requirement of unique names for files in a folder is not a property of the *Folder* schema, it is a property of the *FileSystem* schema.

```
── Folder ──────────────────────────────
│ foldername : NAME
│ files : ℙ File
│ location : FSNODE
│
└────────────────────────────────────────
```

The operating system works with a small number of designated folders where it stores its core files and where third parties might store extensions to the operating system, probably in the form of *libraries*. These folders are *OSFolder* and *OSSysFolder*. There is not made a distinction between these other than that *OSFolder* is more likely to contain program files for interaction with the user, and *OSSysFolder* is more likely to contain library files for process extension.

The *search path* is an ordered set of complete folder names specifying where the operating system should try to locate files when it is instructed to load files referenced by a filename and without a foldername.

$OSFolder : PATHNAME$
$OSSysFolder : PATHNAME$
$SearchPath : \mathrm{seq}\, PATHNAME$

The *FileSystem* schema consists of a *Tree* with nodes of type *FSNODE* and auxiliary functions to retrieve node attributes and to map nodes to *files* and *folders*.

Folders form a tree structure, i.e., starting from the *root* folder at the top, each folder may contain subfolders as well as files. A folder cannot be contained in one of its subfolders, i.e., cycles are not allowed. All file names of files in the same folder must be unique. The same holds for subfolders; all folder names of folders in the same parent folder must be unique. Subfolders and files in the same parent folder may have the same name, though.

The schema includes functions mapping nodes to their corresponding file or folder schema. *nodeName* retrieves the file name or folder name of the corresponding file system tree node. *nodeData* retrieves the content of the file associated with the corresponding file system tree node. *nodeFile* retrieves the file associated with the corresponding file system tree node. *nodeFolder* retrieves the folder associated with the corresponding file system tree node.

subfolders is a function mapping a folder to its contained subfolders. *nodeByFullName* locates the file system tree node referenced by a sequence of folder names and up to one file name. *fileByFullName* fulfils the same purpose, alas resulting in a file schema corresponding to the file system tree node. *folderByFullName* retrieves a folder schema, accordingly. *fullNameByNode* constructs the sequence of folder names (starting with the root folder) and up to one file name corresponding to the specified file system tree node.

FileSystem

$$
\begin{array}{l}
FSTree : Tree[FSNODE] \\
nodeName : FSNODE \pfun NAME \\
nodeData : FSNODE \pfun DATA \\
nodeFile : FSNODE \pfun File \\
nodeFolder : FSNODE \pfun Folder \\
subfolders : Folder \pfun \mathbb{P}\, Folder \\
nodeByFullName : PATHNAME \pfun FSNODE \\
fileByFullName : PATHNAME \pfun File \\
folderByFullName : PATHNAME \pfun Folder \\
fullNameByNode : FSNODE \pfun PATHNAME \\
fileToComponent : File \pfun COMPNAME \\
folderToComponent : Folder \pfun COMPNAME
\end{array}
$$

$$
\begin{array}{l}
\operatorname{dom} nodeName = FSTree.node \\
FSTree.root \notin \operatorname{dom} nodeData \subseteq FSTree.leaf \\
\forall n : FSTree.node \bullet (\forall c, d : FSTree.child (\!|\ \{n\}\ |\!) \mid c \neq d \\
\quad \bullet\ nodeName\ c \neq nodeName\ d) \\
nodeFile = \{n : FSTree.node;\ f : File \mid n \in \operatorname{dom} nodeData \land \\
\quad f.filename = nodeName\ n \land \\
\quad f.content = nodeData\ n \land \\
\quad f.location = n \\
\quad \bullet\ n \mapsto f\} \\
nodeFolder = \{n : FSTree.node;\ fd : Folder \mid n \notin \operatorname{dom} nodeData \land \\
\quad fd.foldername = nodeName\ n \land \\
\quad fd.files = nodeFile (\!|\ (FSTree.child (\!|\ \{n\}\ |\!))\ |\!) \land \\
\quad fd.location = n \\
\quad \bullet\ n \mapsto fd\} \\
subfolders = \{fd : Folder \mid fd.location \in FSTree.node \setminus \operatorname{dom} nodeData \\
\quad \bullet\ fd \mapsto nodeFolder (\!|\ (FSTree.child (\!|\ \{fd.location\}\ |\!))\ |\!)\} \\
nodeByFullName = \{path : \operatorname{seq} FSNODE \mid head\ path = FSTree.root \land \\
\quad (\forall i, j : \operatorname{dom} path \mid j = i + 1 \bullet \\
\quad (path\ i \mapsto path\ j) \in FSTree.child) \\
\quad \bullet\ path \fatsemi nodeName \mapsto last\ path\} \\
fileByFullName = nodeByFullName \fatsemi nodeFile \\
folderByFullName = nodeByFullName \fatsemi nodeFolder \\
fullNameByNode = nodeByFullName^{\sim}
\end{array}
$$

5.2.2.2 Memory

The memory subsystem is responsible for providing and regulating access to storage in main memory.

Memory is divided in *memory areas*. Each such area has a unique identifier – a *handle*

– and stores some content. A memory area is associated with a *process* that is called its *owner*. Memory is private to a process; to share memory between processes, the appropriate access control entries must be present in the access control configuration of the system.

$$
\begin{array}{l}
\textit{MemoryArea} \\
\hline
\textit{handle} : \textit{HANDLE} \\
\textit{content} : \textit{DATA} \\
\textit{owner} : \textit{HANDLE}
\end{array}
$$

The *Memory* schema consists of a set *areas* of the memory areas in use by processes in the system. It provides access via the *areaByHandle* function to a memory area by specifying the area's handle. There is no restriction as regards how much memory can be consumed or is available in total.

$$
\begin{array}{l}
\textit{Memory} \\
\hline
\textit{memoryAreas} : \mathbb{P}\, \textit{MemoryArea} \\
\textit{areaByHandle} : \textit{HANDLE} \pfun \textit{MemoryArea} \\
\textit{memoryAreaToComponent} : \textit{MemoryArea} \pfun \textit{COMPNAME} \\
\hline
\textit{areaByHandle} = \{ h : \textit{HANDLE};\ p : \textit{MemoryArea} \mid \\
\quad p \in \textit{memoryAreas} \land p.\textit{handle} = h \bullet h \mapsto p \} \\
\forall\, m, n : \textit{MemoryArea} \mid m \in \textit{memoryAreas} \land n \in \textit{memoryAreas} \land \\
\quad m \neq n \bullet m.\textit{handle} \neq n.\textit{handle}
\end{array}
$$

5.2.2.3 User and user interface

The user interface subsystem manages the data displayed to the local human user and processes input generated by the user or simulated by processes.

The *UIOutputDevices* and *UIInputDevices* schemas represent the user interface. Each modeled device is identified by a handle. An output device displays the same data until that is changed.

$$
\begin{array}{l}
\textit{UIOutputDevices} \\
\hline
\textit{displayedData} : \textit{HANDLE} \rightarrow \textit{DATA} \\
\textit{uiOutputDeviceToComponent} : \textit{HANDLE} \pfun \textit{COMPNAME}
\end{array}
$$

An input device has a queue of input events.

UIInputDevices

$inputEvents : HANDLE \rightarrow seq\ DATA$
$uiInputDeviceToComponent : HANDLE \nrightarrow COMPNAME$

The complete user interface is the combination of output and input devices. Access restrictions to the user interface may be defined in the system's access control configuration.

UserInterface

UIOutputDevices
UIInputDevices

5.2.2.4 Processes and IPC

The process subsystem manages all running processes, including creation and access control.

An executable *library* consists of executable code, stored in a file. It is referenced by a *handle*. Some parts of a library may contain non-executable data.

LinkedLibraries

$libraryFile : HANDLE \rightarrow File$

A *process* is based on a file (the *program*) whose contents are loaded into main memory and then executed. It is assigned an identifier (*handle*). A process can be extended by runtime modules, e.g., executable libraries. For logging, accounting, and access control purposes, a subject is associated with each process.

The *Processes* schema references the *Memory* and *UI* input/output schemas and consists of a set of *running processes*, a relation associating processes with executable libraries loaded by these processes – *processLibraries*, the set of all loaded libraries in the system, a relation associating processes with the memory areas they have allocated, and a relation associating processes with the input queues they subscribe to.

Processes

$Memory$
$UIOutputDevices$
$UIInputDevices$
$runningProcesses : \mathbb{P}\, HANDLE$
$processFile : HANDLE \rightarrow File$
$processLibraries : HANDLE \leftrightarrow HANDLE$
$processSubject : HANDLE \rightarrow SUBJECT$
$loadedLibraries : \mathbb{P}\, HANDLE$
$processIPCResponse : HANDLE \rightarrow IPCResponse$
$processIPCQueue : HANDLE \rightarrow \mathrm{seq}\, DATA$
$processMemory : HANDLE \leftrightarrow MemoryArea$
$processOutput : HANDLE \leftrightarrow HANDLE$
$processInput : HANDLE \leftrightarrow HANDLE$

$\mathrm{dom}\, processLibraries \subseteq runningProcesses$
$loadedLibraries = \mathrm{ran}\, processLibraries$
$\mathrm{dom}\, processIPCResponse \subseteq runningProcesses$
$\mathrm{dom}\, processMemory \subseteq runningProcesses$
$\mathrm{ran}\, processMemory \subseteq memoryAreas$
$\mathrm{dom}\, processOutput \subseteq runningProcesses$
$\mathrm{ran}\, processOutput \subseteq \mathrm{dom}\, displayedData$
$\mathrm{dom}\, processInput \subseteq runningProcesses$
$\mathrm{ran}\, processInput \subseteq \mathrm{dom}\, inputEvents$

5.2.2.5 Access control

The schema *AccessControlPolicy* represents the access control policy of the system. It references *FileSystem* and *Memory*. The policy is stored as an access matrix, *ACP*, consisting of *ACE*s (access control entries). Half a dozen auxiliary functions map securable system components to objects of the access control policy: *subjectToObject*, *fsnodeToObject*, *fileToObject*, *folderToObject*, *memoryAreaToObject*, *UIAreaToObject*, *processToObject*.

Instead of using the access matrix, a shorter notation for checking access rights is available in the form of *ACL*s (access control lists including all subjects that have access rights for the specified object and their respective rights) and *CAP*s (capability lists including all objects for which the specified subject has access rights and the respective rights). *ACL*s can bee seen as the columns of the access matrix, and *CAP*s represent the rows of the access matrix.

$$
\begin{array}{l}
\textit{AccessControlPolicy} \\
\hline
\textit{FileSystem} \\
\textit{Memory} \\
ACP : ACM \\
subjectToObject : SUBJECT \rightarrowtail OBJECT \\
fsnodeToObject : FSNODE \rightarrowtail OBJECT \\
fileToObject : File \rightarrowtail OBJECT \\
folderToObject : Folder \rightarrowtail OBJECT \\
memoryAreaToObject : MemoryArea \rightarrowtail OBJECT \\
UIOutputDeviceToObject : HANDLE \rightarrowtail OBJECT \\
UIInputDeviceToObject : HANDLE \rightarrowtail OBJECT \\
processToObject : HANDLE \rightarrowtail OBJECT \\
ACL : OBJECT \rightarrow (SUBJECT \rightarrow \mathbb{P}\, ACCESSMODE) \\
CAP : SUBJECT \rightarrow (OBJECT \rightarrow \mathbb{P}\, ACCESSMODE) \\
\hline
fileToObject = \{f : File;\ n : FSTree.node \mid nodeFile\ n = f \\
\quad \bullet f \mapsto fsnodeToObject\ n\} \\
folderToObject = \{fd : Folder;\ n : FSTree.node \mid nodeFolder\ n = fd \\
\quad \bullet fd \mapsto fsnodeToObject\ n\} \\
\langle \mathrm{ran}\, subjectToObject, \mathrm{ran}\, fileToObject, \mathrm{ran}\, folderToObject, \\
\quad \mathrm{ran}\, memoryAreaToObject, \\
\quad \mathrm{ran}\, UIOutputDeviceToObject, \mathrm{ran}\, UIInputDeviceToObject, \\
\quad \mathrm{ran}\, processToObject\rangle \\
\quad \mathrm{partition}\ OBJECT \\
ACL = \{s : SUBJECT;\ o : OBJECT;\ m : \mathbb{P}\, ACCESSMODE \mid \\
\quad ((s, o), m) \in ACP \bullet (o, \\
\qquad \{u : SUBJECT \mid u \in \mathrm{dom}((\mathrm{dom}\, ACP) \rhd \{o\}) \\
\qquad\quad \bullet (u, ACP\ (u, o))\})\} \\
CAP = \{s : SUBJECT;\ o : OBJECT;\ m : \mathbb{P}\, ACCESSMODE \mid \\
\quad ((s, o), m) \in ACP \bullet (s, \\
\qquad \{b : OBJECT \mid b \in \mathrm{ran}(\{s\} \lhd (\mathrm{dom}\, ACP)) \\
\qquad\quad \bullet (b, ACP\ (s, b))\})\}
\end{array}
$$

The schema *SystemHistory* provides a protocol for all relevant actions of an adversary and other processes in the system. These are recorded as an ordered set of *SYSTEMEVENT*s.

$$
\begin{array}{l}
\textit{SystemHistory} \\
\hline
history : \mathrm{seq}\, SYSTEMEVENT
\end{array}
$$

5.2.2.6 Application architecture

Some terms are used throughout the analysis: *OS* (=Operating System), *adversary/attacker* and *victim/process under attack*. These are used to indicate ownership of components. In our specification we introduce them as a given set.

$$OWNER ::= OS \mid ADVERSARY \mid VICTIM \mid UNSPECIFIED$$

An architectural description comprises components and connectors, relating components to each other. A *component* is of a certain *type*, e.g., it can be a data storage component. The component's type determines its importance and behaviour. Components communicate with other components via *ports*. Connectors attach to these ports.

```
__ Component ______________________________
| ports : ℙ PORT
| componentType : COMPTYPE
| owner : OWNER
|__________________________________________
```

Connectors govern the way components interact with each other. Communication depends on the *type* of the connection, e.g., modifying another component's content or invoking another component's method. The ports to which a connector attaches act according to *roles* dependent on the connection's type, e.g., one port can act as a sender, the other as a receiver.

```
__ Connector ______________________________
| roles : ℙ ROLE
| connectionType : CONNTYPE
|__________________________________________
```

The *Configuration* schema represents an architectural description of the system. It assembles instances of *components* and *connectors*. Components and connectors are identified by their names. An *attachment* couples a role instance with a port instance. A role instance is defined by a connection name and a role for one end of the connection. A port instance is defined by a component name and a port of that component. An attachment then maps the connections's roles to the components's ports, e.g., assigning the invoker role to a port of a process component and the invokee role to a port of a file component. *typesAndComponents* and *componentsOfType* are auxiliary relations for retrieving components of a vertain component type.

Configuration

$$
\begin{array}{l}
components : COMPNAME \pfun Component \\
connectors : CONNNAME \pfun Connector \\
attachment : ROLEINST \pfun PORTINST \\
typeComponents : COMPTYPE \leftrightarrow COMPNAME \\
\hline
\forall cn : CONNNAME;\ r : ROLE \mid (cn, r) \in \mathrm{dom}\ attachment \\
\quad \bullet\ cn \in \mathrm{dom}\ connectors \wedge r \in (connectors(cn)).roles \\
\forall cn : COMPNAME;\ p : PORT \mid (cn, p) \in \mathrm{ran}\ attachment \\
\quad \bullet\ cn \in \mathrm{dom}\ components \wedge p \in (components(cn)).ports \\
typeComponents = \{ \\
\quad ct : COMPTYPE;\ cn : COMPNAME;\ cp : Component \mid \\
\quad (cn, cp) \in components \wedge cp.componentType = ct \bullet (ct, cn)\}
\end{array}
$$

5.2.3 Initial state

The *EmptyState* schema defines the initial state of the system in which no components and connectors exist. The system is subsequently populated with files, folders, processes etc.

EmptyState

$$
\begin{array}{l}
FileSystem \\
UIOutputDevices \\
UIInputDevices \\
Processes \\
AccessControlPolicy \\
\hline
FSTree.node \neq \varnothing[FSNODE] \\
\mathrm{ran}\ FSTree.parent = \varnothing[FSNODE] \\
\mathrm{dom}\ \ displayedData = \varnothing[HANDLE] \\
\mathrm{dom}\ \ inputEvents = \varnothing[HANDLE] \\
runningProcesses = \varnothing[HANDLE]
\end{array}
$$

Initial states for the several sub systems of our abstracted operating system are defined by their own schemas. For every object in the operating system model that belongs to our application, there must be a corresponding component or connector in the architecture model.

InitialFileSystem is the initial configuration of the file system. For every file and folder there is a corresponding *data* or *container* component in the architecture model.

InitialFileSystem

FileSystem
Configuration

$$
\begin{array}{l}
\forall fileNode : FSNODE \mid fileNode \in FSTree.node \land fileNode \in \mathrm{dom}\ nodeFile \bullet \\
\quad \exists compname : COMPNAME \mid compname \in \mathrm{dom}\ components \land \\
\quad\quad (components\ compname).componentType = CTDATA \bullet \\
\quad \exists contcompname : COMPNAME \mid contcompname \in \mathrm{dom}\ components \land \\
\quad\quad (components\ contcompname).componentType = CTDATA \bullet \\
\quad \exists connname : CONNNAME \mid connname \in \mathrm{dom}\ connectors \land \\
\quad\quad (connectors\ connname).connectionType = CNCONTAINEDBY \bullet \\
\quad \exists compport : PORT \bullet \exists contport : PORT \bullet \\
\quad\quad ((connname, ROCONTAINER), (contcompname, contport)) \in \\
\quad\quad\quad attachment \land \\
\quad\quad ((connname, ROCONTAINED), (compname, compport)) \in \\
\quad\quad\quad attachment \land \\
\quad\quad ((nodeFile\ fileNode), compname) \in fileToComponent \\
\forall folderNode : FSNODE \mid folderNode \in FSTree.node \land \\
\quad folderNode \in \mathrm{dom}\ nodeFolder \bullet \\
\quad \exists contcompname : COMPNAME \mid contcompname \in \mathrm{dom}\ components \land \\
\quad\quad (components\ contcompname).componentType = CTDATA \bullet \\
\quad \exists compname : COMPNAME \mid compname \in \mathrm{dom}\ components \land \\
\quad\quad (components\ compname).componentType = CTDATA \bullet \\
\quad \exists connname : CONNNAME \mid connname \in \mathrm{dom}\ connectors \land \\
\quad\quad (connectors\ connname).connectionType = CNCONTAINEDBY \bullet \\
\quad \exists contport : PORT \bullet \exists compport : PORT \bullet \\
\quad\quad ((connname, ROCONTAINER), (contcompname, contport)) \in \\
\quad\quad\quad attachment \land \\
\quad\quad ((connname, ROCONTAINED), (compname, compport)) \in \\
\quad\quad\quad attachment \land \\
\quad\quad ((nodeFolder\ folderNode), compname) \in folderToComponent \\
\forall f, fd : FSNODE \mid (f, fd) \in FSTree.child \bullet \exists connname : CONNNAME \bullet \\
\quad (connectors\ connname).connectionType = CNCONTAINEDBY \land \\
\quad \exists contcompname : COMPNAME \bullet \exists contport : PORT \bullet \\
\quad\quad contport \in (components\ contcompname).ports \land \\
\quad \exists compname : COMPNAME \bullet \exists compport : PORT \bullet \\
\quad\quad compport \in (components\ compname).ports \land \\
\quad\quad ((connname, ROCONTAINER), (contcompname, contport)) \in \\
\quad\quad\quad attachment \land \\
\quad\quad ((connname, ROCONTAINED), (compname, compport)) \in \\
\quad\quad\quad attachment
\end{array}
$$

InitialMemory is the initial configuration of the memory subsystem. For every memory area there is a corresponding *data* component in the architecture model.

$$
\begin{array}{l}
\textit{InitialMemory} \\
\hline
\textit{Memory} \\
\textit{Configuration} \\
\hline
\forall m : \textit{MemoryArea} \bullet \\
\quad m \in \textit{memoryAreas} \land \\
\quad \exists \textit{compname} : \textit{COMPNAME} \bullet \\
\quad\quad \textit{compname} \in \operatorname{dom} \textit{components} \land \\
\quad\quad (\textit{components compname}).\textit{componentType} = \textit{CTDATA} \land \\
\quad\quad (m, \textit{compname}) \in \textit{memoryAreaToComponent}
\end{array}
$$

InitialUI is the initial configuration of the user interface devices. For every user interface device there is a corresponding *UI output* or *input* component in the architecture model.

$$
\begin{array}{l}
\textit{InitialUI} \\
\hline
\textit{UIOutputDevices} \\
\textit{UIInputDevices} \\
\textit{Configuration} \\
\hline
\forall u : \textit{HANDLE} \mid \\
\quad u \in \operatorname{dom} \textit{displayedData} \bullet \\
\quad \exists \textit{compname} : \textit{COMPNAME} \bullet \\
\quad\quad (\textit{compname} \in \operatorname{dom} \textit{components} \land \\
\quad\quad (\textit{components compname}).\textit{componentType} = \textit{CTUIOUTPUT} \land \\
\quad\quad (u, \textit{compname}) \in \textit{uiOutputDeviceToComponent}) \\
\forall u : \textit{HANDLE} \mid \\
\quad u \in \operatorname{dom} \textit{inputEvents} \bullet \\
\quad \exists \textit{compname} : \textit{COMPNAME} \bullet \\
\quad\quad (\textit{compname} \in \operatorname{dom} \textit{components} \land \\
\quad\quad (\textit{components compname}).\textit{componentType} = \textit{CTUIINPUT} \land \\
\quad\quad (u, \textit{compname}) \in \textit{uiInputDeviceToComponent})
\end{array}
$$

ProcessLibraryCorrespondence defines the relation *correspondingLibraries* that connects components representing processes and the libraries linked to them.

$$
\begin{array}{l}
\textit{ProcessLibraryCorrespondence} \\
\hline
\mathit{Configuration} \\
\mathit{FileSystem} \\
\mathit{Processes} \\
\mathit{correspondingLibraries} : \mathit{COMPNAME} \leftrightarrow \mathit{HANDLE} \\
\hline
\mathit{correspondingLibraries} = \{\mathit{compname} : \mathit{COMPNAME};\ \mathit{lib} : \mathit{HANDLE} \mid \\
\quad \exists\, \mathit{libcompname} : \mathit{COMPNAME} \bullet \\
\quad \exists\, \mathit{fn} : \mathit{FSNODE} \bullet \\
\quad (\mathit{fileToComponent}\ (\mathit{nodeFile}\ \mathit{fn})) = \mathit{libcompname} \land \\
\quad \exists\, \mathit{procport} : \mathit{PORT};\ \mathit{libport} : \mathit{PORT} \bullet \\
\quad \exists\, \mathit{connname} : \mathit{CONNNAME} \bullet \\
\quad (\mathit{connectors}\ \mathit{connname}).\mathit{connectionType} = \mathit{CNLINKEDEXEC} \land \\
\quad ((\mathit{connname}, \mathit{ROLINKEXECUTOR}), (\mathit{compname}, \mathit{procport})) \in \mathit{attachment} \land \\
\quad ((((\mathit{connname}, \mathit{ROLINKEXECUTEDSTATIC}), \\
\qquad (\mathit{libcompname}, \mathit{libport})) \in \mathit{attachment}) \lor \\
\quad (((\mathit{connname}, \mathit{ROLINKEXECUTEDDYNAMIC}), \\
\qquad (\mathit{libcompname}, \mathit{libport})) \in \mathit{attachment})) \bullet \\
\quad (\mathit{compname}, \mathit{lib})\}
\end{array}
$$

ProcessMemoryCorrespondence defines the relation *correspondingMemoryAreas* that connects components representing processes and the memory areas used by them.

$$
\begin{array}{l}
\textit{ProcessMemoryCorrespondence} \\
\hline
\mathit{Configuration} \\
\mathit{FileSystem} \\
\mathit{Memory} \\
\mathit{Processes} \\
\mathit{correspondingMemoryAreas} : \mathit{COMPNAME} \leftrightarrow \mathit{MemoryArea} \\
\hline
\mathit{correspondingMemoryAreas} = \{\mathit{compname} : \mathit{COMPNAME};\ \mathit{ma} : \mathit{MemoryArea} \mid \\
\quad \exists\, \mathit{macompname} : \mathit{COMPNAME} \bullet \\
\quad \exists\, \mathit{fn} : \mathit{FSNODE} \bullet \\
\quad (\mathit{fileToComponent}\ (\mathit{nodeFile}\ \mathit{fn})) = \mathit{macompname} \land \\
\quad \exists\, \mathit{procport} : \mathit{PORT};\ \mathit{maport} : \mathit{PORT} \bullet \\
\quad \exists\, \mathit{connname} : \mathit{CONNNAME} \bullet \\
\quad (\mathit{connectors}\ \mathit{connname}).\mathit{connectionType} = \mathit{CNDATATRANSFER} \land \\
\quad ((\mathit{connname}, \mathit{RODATATARGET}), (\mathit{compname}, \mathit{procport})) \in \mathit{attachment} \land \\
\quad ((((\mathit{connname}, \mathit{RODATASOURCE}), (\mathit{macompname}, \mathit{maport})) \in \mathit{attachment}) \lor \\
\quad (((\mathit{connname}, \mathit{ROAUTHENTICATEDDATASOURCE}), \\
\qquad (\mathit{macompname}, \mathit{maport})) \in \mathit{attachment})) \bullet \\
\quad (\mathit{compname}, \mathit{ma})\}
\end{array}
$$

ProcessUIOutputCorrespondence defines the relation *correspondingUIOutput* that connects components representing processes and the UI output devices used by them.

$$
\begin{array}{l}
\textit{ProcessUIOutputCorrespondence} \\
\hline
\textit{Configuration} \\
\textit{UserInterface} \\
\textit{Processes} \\
\textit{correspondingUIOutput} : \textit{COMPNAME} \leftrightarrow \textit{HANDLE} \\
\hline
\textit{correspondingUIOutput} = \{\textit{compname} : \textit{COMPNAME};\ \textit{uio} : \textit{HANDLE} \mid \\
\quad \exists\, \textit{uiocompname} : \textit{COMPNAME} \bullet \\
\quad (\textit{uiOutputDeviceToComponent}\ \textit{uio}) = \textit{uiocompname} \land \\
\quad \exists\, \textit{procport} : \textit{PORT};\ \textit{uioport} : \textit{PORT} \bullet \\
\quad \exists\, \textit{connname} : \textit{CONNNAME} \bullet \\
\quad (\textit{connectors}\ \textit{connname}).\textit{connectionType} = \textit{CNDATATRANSFER} \land \\
\quad ((((\textit{connname}, \textit{RODATASOURCE}), (\textit{compname}, \textit{procport})) \in \textit{attachment}) \lor \\
\quad ((\textit{connname}, \textit{ROAUTHENTICATEDDATASOURCE}), \\
\qquad (\textit{compname}, \textit{procport})) \in \textit{attachment}) \land \\
\quad ((((\textit{connname}, \textit{RODATATARGET}), (\textit{uiocompname}, \textit{uioport})) \in \textit{attachment}) \lor \\
\quad (((\textit{connname}, \textit{ROAUTHENTICATEDDATATARGET}), \\
\qquad (\textit{uiocompname}, \textit{uioport})) \in \textit{attachment})) \bullet \\
\quad (\textit{compname}, \textit{uio})\} \\
\hline
\end{array}
$$

ProcessUIInputCorrespondence defines the relation *correspondingUIInput* that connects components representing processes and the UI input devices used by them.

$$
\begin{array}{l}
\textit{ProcessUIInputCorrespondence} \\
\hline
\textit{Configuration} \\
\textit{UserInterface} \\
\textit{Processes} \\
\textit{correspondingUIInput} : \textit{COMPNAME} \leftrightarrow \textit{HANDLE} \\
\hline
\textit{correspondingUIInput} = \{\textit{compname} : \textit{COMPNAME};\ \textit{uii} : \textit{HANDLE} \mid \\
\quad \exists\, \textit{uiicompname} : \textit{COMPNAME} \bullet \\
\quad (\textit{uiInputDeviceToComponent}\ \textit{uii}) = \textit{uiicompname} \land \\
\quad \exists\, \textit{procport} : \textit{PORT};\ \textit{uiiport} : \textit{PORT} \bullet \\
\quad \exists\, \textit{connname} : \textit{CONNNAME} \bullet \\
\quad (\textit{connectors}\ \textit{connname}).\textit{connectionType} = \textit{CNDATATRANSFER} \land \\
\quad ((((\textit{connname}, \textit{RODATATARGET}), (\textit{compname}, \textit{procport})) \in \textit{attachment}) \lor \\
\quad ((\textit{connname}, \textit{ROAUTHENTICATEDDATATARGET}), \\
\qquad (\textit{compname}, \textit{procport})) \in \textit{attachment}) \land \\
\quad ((((\textit{connname}, \textit{RODATASOURCE}), (\textit{uiicompname}, \textit{uiiport})) \in \textit{attachment}) \lor \\
\quad (((\textit{connname}, \textit{ROAUTHENTICATEDDATASOURCE}), \\
\qquad (\textit{uiicompname}, \textit{uiiport})) \in \textit{attachment})) \bullet \\
\quad (\textit{compname}, \textit{uii})\} \\
\hline
\end{array}
$$

The initially running processes are given by *InitialProcesses*. Only processes executed by the local human user are considered. Processes launched automatically, e.g., by a mechanism of the operating system, are left to future work. For every process all linked libraries and allocated memory are mapped to corresponding components in the architectural description, as well as output and input devices of the user interface.

$$
\begin{array}{l}
\textbf{InitialProcesses} \\
\hline
Processes \\
FileSystem \\
UserInterface \\
Configuration \\
ProcessLibraryCorrespondence \\
ProcessMemoryCorrespondence \\
ProcessUIOutputCorrespondence \\
ProcessUIInputCorrespondence \\
\hline
\forall proc : HANDLE \mid \\
\quad proc \in runningProcesses \bullet \\
\quad \exists compname : COMPNAME \bullet \\
\quad compname \in \mathrm{dom}\ components \land \\
\quad fileToComponent\ (processFile\ proc) = compname \land \\
\quad processSubject\ proc = SUBJECT_VICTIM \land \\
\qquad (\exists usercompname : COMPNAME \bullet \\
\qquad \exists procport : PORT;\ userport : PORT \bullet \\
\qquad (components\ usercompname).componentType = CTHUMANUSER \land \\
\qquad \exists connname : CONNNAME \bullet \\
\qquad (connectors\ connname).connectionType = CNEXECUTE \land \\
\qquad ((connname, ROEXECUTOR), (usercompname, userport)) \in attachment \land \\
\qquad ((connname, ROEXECUTED), (compname, procport)) \in attachment) \land \\
\quad (\forall lib : HANDLE \bullet \\
\qquad lib \in \mathrm{ran}\ processLibraries \Rightarrow \\
\qquad (compname, lib) \in correspondingLibraries) \land \\
\quad (\forall ma : MemoryArea \bullet \\
\qquad ma \in \mathrm{ran}\ processMemory \Rightarrow \\
\qquad (compname, ma) \in correspondingMemoryAreas) \land \\
\quad (\forall uio : HANDLE \bullet \\
\qquad uio \in \mathrm{ran}\ processOutput \Rightarrow \\
\qquad (compname, uio) \in correspondingUIOutput) \land \\
\quad (\forall uii : HANDLE \bullet \\
\qquad uii \in \mathrm{ran}\ processInput \Rightarrow \\
\qquad (compname, uii) \in correspondingUIInput)
\end{array}
$$

The initial access matrix is built up from a set of access control entries ($object \times \times subject \times access\ mode$). For every securable object and for every subject there is a corresponding component in the architectural description. Access modes are determined

by corresponding connector types between components connected to *subject type* components.

InitialConfigurationACE

$$
\begin{array}{l}
Configuration \\
FileSystem \\
Memory \\
UserInterface \\
AccessControlPolicy \\
InitialACE : (OBJECT \times SUBJECT) \rightarrow ACCESSMODE \\
\hline
InitialACE = \{ \\
\quad objcompname : COMPNAME;\ subjcompname : COMPNAME; \\
\quad obj : OBJECT;\ subj : SUBJECT; \\
\quad fn : FSNODE;\ ma : MemoryArea; \\
\quad uio : HANDLE;\ uii : HANDLE; \\
\quad connname : CONNNAME; \\
\quad objport : PORT;\ subjport : PORT \mid \\
\qquad ((fn \in FSTree.node \land \\
\qquad\quad (objcompname = (fileToComponent\ (nodeFile\ fn))) \land \\
\qquad\quad obj = fsnodeToObject\ fn) \lor \\
\qquad (fn \in FSTree.node \land \\
\qquad\quad (objcompname = (folderToComponent\ (nodeFolder\ fn))) \land \\
\qquad\quad obj = fsnodeToObject\ fn) \lor \\
\qquad (ma \in memoryAreas \land \\
\qquad\quad (objcompname = (memoryAreaToComponent\ ma)) \land \\
\qquad\quad obj = memoryAreaToObject\ ma) \lor \\
\qquad (uio \in \operatorname{dom} uiOutputDeviceToComponent \land \\
\qquad\quad (objcompname = (uiOutputDeviceToComponent\ uio)) \land \\
\qquad\quad obj = UIOutputDeviceToObject\ uio) \lor \\
\qquad (uii \in \operatorname{dom} uiInputDeviceToComponent \land \\
\qquad\quad (objcompname = (uiInputDeviceToComponent\ uii)) \land \\
\qquad\quad obj = UIInputDeviceToObject\ uii)) \land \\
\qquad (components\ subjcompname).componentType \in \\
\qquad\quad subjectComponentTypes \land \\
\qquad subj = (subjectComponentTypeToSUBJECT\ (\\
\qquad\quad components\ subjcompname).componentType) \land \\
\qquad (connectors\ connname).connectionType \in accessConnectorTypes \land \\
\qquad ((connname, ROOBJECT), (objcompname, objport)) \in attachment \land \\
\qquad ((connname, ROSUBJECT), (subjcompname, subjport)) \in attachment \bullet \\
\quad ((obj, subj), (accessConnectorTypeToACCESSMODE\ (\\
\qquad connectors\ connname).connectionType))\}
\end{array}
$$

Combining the initial access control entries leads to the initial access matrix ACP.

$$
\begin{array}{l}
\textit{InitialAccessControlPolicy} \\
\hline
\textit{InitialConfigurationACE} \\
\textit{AccessControlPolicy} \\
\hline
\textit{ACP} = \{ \\
\quad \textit{obj} : \textit{OBJECT};\ \textit{subj} : \textit{SUBJECT} \bullet \\
\quad ((\textit{subj}, \textit{obj}), \textit{InitialACE}(\!|\ \{(\textit{obj}, \textit{subj})\}\ |\!)\ ) \\
\}
\end{array}
$$

All components in the architectural description shall be covered by files, folders, memory etc. in the model. This is expressed with the *AllComponentsCoveredSchema*.

$$
\begin{array}{l}
\textit{AllComponentsCovered} \\
\hline
\textit{Configuration} \\
\textit{FileSystem} \\
\textit{Memory} \\
\textit{UserInterface} \\
\hline
\textit{typeComponents}(\!|\ \textit{MappableComponentTypes}\ |\!) = \\
\quad (\operatorname{ran} \textit{fileToComponent}) \cup \\
\quad (\operatorname{ran} \textit{folderToComponent}) \cup \\
\quad (\operatorname{ran} \textit{memoryAreaToComponent}) \cup \\
\quad (\operatorname{ran} \textit{uiOutputDeviceToComponent}) \cup \\
\quad (\operatorname{ran} \textit{uiInputDeviceToComponent})
\end{array}
$$

The initial state is composed of the initial states of the subsystems and is constrained by that there be an entity in the model for every relevant component in the architectural description.

$$
\begin{array}{l}
\textit{InitialStateCorrespondence} \\
\hline
\textit{InitialFileSystem} \\
\textit{InitialMemory} \\
\textit{InitialUI} \\
\textit{InitialProcesses} \\
\textit{InitialAccessControlPolicy} \\
\textit{AllComponentsCovered}
\end{array}
$$

5.2.4 Operations

When using schemas, we adhere to the convention that Δ*schema name* denotes a schema where state changes may take place.

$\Delta SchemaName$

$SchemaName$
$SchemaName'$

In a similar way we use the established convention to write Ξ*schema name* for schema inclusion where the state is left unchanged.

$\Xi SchemaName$

$\Delta SchemaName$

$\theta SchemaName' = \theta SchemaName$

The usual naming style for input variables is to decorate them with a question mark (?), and to decorate output variables with an exclamation mark (!).

5.2.4.1 File system

The file system supports creation, reading, modification, appending, and deletion of files and folders.

The *CreateFile* schema includes the three schemas *FileSystem*, *AccessControlPolicy*, and *SystemHistory*. File creation requires provision of the name of the file to be created, the (existing) folder in which it shall be placed, the owner of the newly created file, and the access rights for that owner to enable the creation of an ACE of the new file.

Precondition for this operation is that the folder exists. If the precondition is met, a new file with content *CONTENT_EMPTY* is created and assigned an unused *FSNODE* in the file system. It is placed in the file system tree with the specified folder as its parent node. The access control configuration is updated with an ACE for the new file. File creation is logged in the system history.

$$
\begin{array}{l}
\textit{CreateFile} \\
\hline
\Delta \textit{FileSystem} \\
\Delta \textit{AccessControlPolicy} \\
\Delta \textit{SystemHistory} \\
\textit{newFilename}? : \textit{PATHNAME} \\
\textit{newFileAccessMode}? : \mathbb{P}(\textit{SUBJECT} \times \mathbb{P}\, \textit{ACCESSMODE}) \\
\textit{requestingSubject}? : \textit{SUBJECT} \\
\hline
(\textit{nodeByFullName}\ (\textit{front}\ \textit{newFilename}?)) \in \textit{FSTree.node} \setminus \operatorname{dom} \textit{nodeData} \\
\exists f : \textit{File} \mid \\
\quad f.\textit{filename} = \textit{last}\ \textit{newFilename}? \land \\
\quad f.\textit{content} \in \textit{CONTENT_EMPTY} \land \\
\quad f.\textit{location} \notin \textit{FSTree.node} \land \\
\quad \textit{ACCESS_MODIFY} \in \\
\qquad ((\textit{ACL}\ (\textit{fsnodeToObject}\ (\textit{FSTree.parent}\ f.\textit{location})))\ \textit{requestingSubject}?) \\
\quad \bullet\ \textit{FSTree}'.\textit{node} = \textit{FSTree.node} \cup \{f.\textit{location}\} \land \\
\qquad (\textit{FSTree}').\textit{parent} = \textit{FSTree.parent} \oplus \\
\qquad\quad \{f.\textit{location} \mapsto \textit{nodeByFullName}\ (\textit{front}\ \textit{newFilename}?)\} \land \\
\qquad \textit{nodeName}' = \textit{nodeName} \oplus \{f.\textit{location} \mapsto f.\textit{filename}\} \land \\
\qquad \textit{nodeData}' = \textit{nodeData} \oplus \{f.\textit{location} \mapsto f.\textit{content}\} \land \\
\qquad \textit{ACP}' = \textit{ACP} \oplus \{s : \textit{SUBJECT};\ m : \mathbb{P}\, \textit{ACCESSMODE} \mid \\
\qquad\quad (s, m) \in \textit{newFileAccessMode}? \bullet ((s, \textit{fileToObject}\ f), m)\} \land \\
\qquad \textit{history}' = \textit{history} \frown \\
\qquad\quad \{1 \mapsto ((((\textit{subjectComponentTypeToSUBJECT})^{\sim})\textit{requestingSubject}?), \\
\qquad\quad \textit{fileToComponent}\ f, \textit{EVENT_ACCESS_CREATE})\}
\end{array}
$$

The *ReadFile* schema includes the *FileSystem* and *AccessControlPolicy* schemas without state changes, and the *SystemHistory* schema. Reading a file's content requires provision of the name of the file to be read, and the subject attempting to access the data. Read content is stored in the *filecontent*! schema variable.

Precondition for this operation is that the file exists and that the access control configuration includes *observe* access for the requesting subject. The file access is logged in the system history.

$$
\begin{array}{l}
\textit{ReadFile} \\
\hline
\Xi \textit{FileSystem} \\
\Xi \textit{AccessControlPolicy} \\
\Delta \textit{SystemHistory} \\
\textit{filename}? : \textit{PATHNAME} \\
\textit{filecontent}! : \textit{DATA} \\
\textit{requestingSubject}? : \textit{SUBJECT} \\
\hline
\exists f : \textit{File} \bullet \\
\quad f = \textit{nodeFile}\ (\textit{nodeByFullName}\ \textit{filename}?) \land \\
\quad f.\textit{content} = \textit{filecontent}! \land \\
\quad \textit{ACCESS_OBSERVE} \in ((\textit{ACL}\ (\textit{fileToObject}\ f))\ \textit{requestingSubject}?) \land \\
\quad \textit{history}' = \textit{history} \frown \\
\qquad \{1 \mapsto ((((\textit{subjectComponentTypeToSUBJECT})^{\sim})\textit{requestingSubject}?), \\
\qquad \textit{fileToComponent}\ f, \textit{EVENT_ACCESS_OBSERVE})\}
\end{array}
$$

The *UpdateFile* schema includes the three schemas *FileSystem*, *AccessControlPolicy*, and *SystemHistory*. Updating a file's content requires provision of the name of the file to be updated, the new content of the file, and the subject attempting to update the file's content.

Precondition for this operation is that the file exists and that the access control configuration includes *modify* access for the requesting subject. The file access is logged in the system history.

$$
\begin{array}{l}
\textit{UpdateFile} \\
\hline
\Delta \textit{FileSystem} \\
\Xi \textit{AccessControlPolicy} \\
\Delta \textit{SystemHistory} \\
\textit{filename}? : \textit{PATHNAME} \\
\textit{filecontent}? : \textit{DATA} \\
\textit{requestingSubject}? : \textit{SUBJECT} \\
\hline
\exists f : \textit{File} \mid \\
\quad f = \textit{nodeFile}\ (\textit{nodeByFullName}\ \textit{filename}?) \land \\
\quad f.\textit{content} = \textit{filecontent}? \land \\
\quad \textit{ACCESS_MODIFY} \in ((\textit{ACL}\ (\textit{fileToObject}\ f))\ \textit{requestingSubject}?) \\
\qquad \bullet \textit{nodeData}' = \textit{nodeData} \oplus \{f.\textit{location} \mapsto f.\textit{content}\} \land \\
\qquad \textit{history}' = \textit{history} \frown \\
\qquad\quad \{1 \mapsto ((((\textit{subjectComponentTypeToSUBJECT})^{\sim})\textit{requestingSubject}?), \\
\qquad\quad \textit{fileToComponent}\ f, \textit{EVENT_ACCESS_MODIFY})\}
\end{array}
$$

The *RenameFile* schema includes the three schemas *FileSystem*, *AccessControlPolicy*, and *SystemHistory*. Renaming a file requires provision of the name of the file to be renamed, the desired new name of the file, and the subject attempting to rename the file.

Precondition for this operation is that the file referenced by the old name exists and that the access control configuration includes *modify* access for the requesting subject on the folder where the file is located. Renaming is logged in the system history. There may be global constraints defined in the *FileSystem* schema that restrict names depending on other files residing in the same folder. A file can only be given a new name by *RenameFile*. It cannot be moved to a different folder by that operation. For this purpose, *MoveFile* should be used instead.

$$
\begin{array}{l}
\textbf{RenameFile} \\
\Delta FileSystem \\
\Xi AccessControlPolicy \\
\Delta SystemHistory \\
oldFilename? : PATHNAME \\
newFilename? : NAME \\
requestingSubject? : SUBJECT \\
\hline
\exists f : File \mid \\
\quad f = nodeFile\ (nodeByFullName\ oldFilename?) \wedge \\
\quad ACCESS_MODIFY \in \\
\qquad ((ACL\ (fsnodeToObject\ (FSTree.parent\ f.location)))\ requestingSubject?) \\
\qquad \bullet\ nodeName' = nodeName \oplus \{f.location \mapsto newFilename?\} \wedge \\
\qquad history' = history \frown \\
\qquad\quad \{1 \mapsto ((((subjectComponentTypeToSUBJECT)^{\sim})requestingSubject?), \\
\qquad\quad fileToComponent\ f, EVENT_ACCESS_RENAME)\}
\end{array}
$$

The *AppendFile* schema represents the operation of updating a file's content by adding data to it at the end. It includes the *UpdateFile* schema since appending is a special case of updating. Appending to a file requires provision of the file name, the new content, and the subject attempting to append to the file's content. The format of the data is not relevant; whether the new file content is result of an *append* operation is determined by membership in the *appended* relation.

Precondition for this operation is – in accordance with the precondition for the *UpdateFile* operation – that the file exists and that the access control configuration includes *modify* access for the requesting subject. The file access is logged in the system history.

$$
\begin{array}{l}
\textbf{AppendFile} \\
UpdateFile \\
\hline
filecontent? \in appended\ (\!|\ \{nodeData\ (nodeByFullName\ filename?)\}\ |\!)
\end{array}
$$

The *CopyFile* schema represents the operation of copying a file. Copying a file can be composed of creating an empty new file, reading from the source file, and updating the new target file with the source file's content. Copying a file requires provision of the names of the source file, *filename?*, and the target file, *newFilename?*, as well as the

subject attempting to copy the file.

Precondition for this operation is – in accordance with the preconditions for the *CreateFile*, *ReadFile*, and *UpdateFile* operations – that the file exists and that the access control configuration includes *observe* and *modify* access for the requesting subject. Access permissions for the target file are the same as those for the requesting subject for the source file. The file access is logged in the system history.

$$
\begin{aligned}
&\mathit{CopyFile} == [\mathit{CreateFile};\ \mathit{ReadFile} \mid \\
&\quad \mathit{newFileAccessMode}? = \mathit{ACL}\ (\mathit{fileToObject}\ (\mathit{fileByFullName}\ \mathit{filename}?)) \\
&\qquad] \land \\
&\quad \mathit{UpdateFile}[\mathit{newFilename}?/\mathit{filename}?, \mathit{filecontent}!/\mathit{filecontent}?]
\end{aligned}
$$

The *DeleteFile* schema includes the three schemas *FileSystem*, *AccessControlPolicy*, and *SystemHistory*. Deleting a file requires provision of the name of the file to be deleted and the subject attempting to delete the file.

Precondition for this operation is that the file exists and that the access control configuration includes *delete* access for the requesting subject. Deletion is logged in the system history. After deletion the file and its content are no longer available in the system.

$$
\begin{aligned}
&\textbf{DeleteFile} \\
&\Delta \mathit{FileSystem} \\
&\Delta \mathit{AccessControlPolicy} \\
&\Delta \mathit{SystemHistory} \\
&\mathit{filename}? : \mathit{PATHNAME} \\
&\mathit{requestingSubject}? : \mathit{SUBJECT} \\
&\hline \\
&\exists f : \mathit{File} \mid \\
&\quad f = \mathit{nodeFile}\ (\mathit{nodeByFullName}\ \mathit{filename}?) \land \\
&\quad f.\mathit{location} = \mathit{nodeByFullName}\ \mathit{filename}? \land \\
&\quad \mathit{ACCESS_MODIFY} \in \\
&\qquad ((\mathit{ACL}\ (\mathit{fsnodeToObject}\ (\mathit{FSTree.parent}\ f.\mathit{location})))\ \mathit{requestingSubject}?) \land \\
&\quad \mathit{ACCESS_DELETE} \in ((\mathit{ACL}\ (\mathit{fileToObject}\ f))\ \mathit{requestingSubject}?) \\
&\qquad \bullet\ \mathit{nodeData}' = \{f.\mathit{location}\} \mathbin{⩤} \mathit{nodeData} \land \\
&\qquad \mathit{nodeName}' = \{f.\mathit{location}\} \mathbin{⩤} \mathit{nodeName} \land \\
&\qquad \mathit{FSTree}'.\mathit{parent} = \{f.\mathit{location}\} \mathbin{⩤} \mathit{FSTree.parent} \land \\
&\qquad \mathit{FSTree}'.\mathit{node} = \mathit{FSTree.node} \setminus \{f.\mathit{location}\} \land \\
&\qquad \mathit{ACP}' = \{s : \mathit{SUBJECT};\ o : \mathit{OBJECT};\ m : \mathbb{P}\,\mathit{ACCESSMODE} \mid \\
&\qquad\quad ((s, o), m) \in \mathit{ACP} \land o \neq \mathit{fileToObject}\ f \bullet ((s, o), m)\} \land \\
&\qquad \mathit{history}' = \mathit{history} ^\frown \\
&\qquad\quad \{1 \mapsto ((((\mathit{subjectComponentTypeToSUBJECT})^\sim)\mathit{requestingSubject}?), \\
&\qquad\quad \mathit{fileToComponent}\ f, \mathit{EVENT_ACCESS_DELETE})\}
\end{aligned}
$$

The *MoveFile* schema represents the operation of moving a file to a new location. Moving a file can be composed of copying the source file to the new destination and deleting the source file. Moving requires provision of the names of the source file, *filename*?, the target file, *newFilename*?, and the subject attempting to move the file.

Precondition for this operation is – in accordance with the preconditions for the *CopyFile* and *DeleteFile* operations – that the file exists and that the access control configuration includes *observe*, *modify* and *delete* access for the requesting subject. Access permissions for the target file are the same as those for the requesting subject for the source file. The file access is logged in the system history.

$$MoveFile == CopyFile \land DeleteFile$$

The *CreateFolder* schema includes the three schemas *FileSystem*, *AccessControlPolicy*, and *SystemHistory*. Folder creation requires provision of the name of the folder to be created, the (existing) folder under which it shall be placed, the owner of the newly created folder, and the access rights for that owner to enable the creation of an ACE of the new folder.

Precondition for this operation is that the parent folder exists. If the precondition is met, a new folder is created and assigned an unused *FSNODE* in the file system. It is placed in the file system tree with the specified *parentfolder*? as its parent node. The access control configuration is updated with an ACE for the new folder. Folder creation is logged in the system history.

CreateFolder

$$\begin{array}{l}
\Delta FileSystem \\
\Delta AccessControlPolicy \\
\Delta SystemHistory \\
parentfolder? : PATHNAME \\
newFoldername? : NAME \\
newFolderOwner? : SUBJECT \\
newFolderAccessMode? : \mathbb{P}(SUBJECT \times \mathbb{P}\, ACCESSMODE) \\
requestingSubject? : SUBJECT \\
\hline
nodeByFullName\ parentfolder? \in FSTree.node \setminus \mathrm{dom}\, nodeData \\
\exists\, fd : Folder \mid \\
\quad fd.foldername = newFoldername? \land \\
\quad fd.location \notin FSTree.node \land \\
\quad ACCESS_MODIFY \in \\
\qquad ((ACL\ (fsnodeToObject\ (nodeByFullName\ parentfolder?)))\ requestingSubject?) \\
\qquad \bullet\ FSTree'.node = FSTree.node \cup \{fd.location\} \land \\
\qquad FSTree'.parent = FSTree.parent \oplus \\
\qquad\quad \{fd.location \mapsto nodeByFullName\ parentfolder?\} \land \\
\qquad nodeName' = nodeName \oplus \{fd.location \mapsto fd.foldername\} \land \\
\qquad nodeData' = \{fd.location\} \lhd nodeData \land \\
\qquad ACP' = ACP \oplus \{s : SUBJECT;\ m : \mathbb{P}\, ACCESSMODE \mid \\
\qquad\quad (s, m) \in newFolderAccessMode? \bullet ((s, folderToObject\ fd), m)\} \land \\
\qquad history' = history \frown \\
\qquad\quad \{1 \mapsto ((((subjectComponentTypeToSUBJECT)^{\sim})requestingSubject?), \\
\qquad\quad folderToComponent\ fd, EVENT_ACCESS_CREATE)\}
\end{array}$$

The *DeleteFolder* schema includes the three schemas *FileSystem*, *AccessControlPolicy*, and *SystemHistory*. Folder deletion requires provision of the name of the folder to be deleted and the subject attempting to delete the folder.

Precondition for this operation is that the folder exists and that the access control configuration includes *delete* access for the requesting subject. It is only possible to delete a folder if it does not contain any files or subfolders. If the precondition is met, the *FSNODE* associated with the folder is removed from the file system tree and is no longer available in the system. Folder deletion is logged in the system history.

(As an aside, to model the behaviour of the Unix `rm -r` command, one could recursively apply *DeleteFile* and *DeleteFolder* to all subfolders and files contained in the folder to be deleted.)

$$
\begin{array}{l}
\textbf{DeleteFolder} \\
\hline
\Delta FileSystem \\
\Delta AccessControlPolicy \\
\Delta SystemHistory \\
foldername? : PATHNAME \\
requestingSubject? : SUBJECT \\
\hline
\exists fd : Folder \mid \\
\quad fd = nodeFolder\ (nodeByFullName\ foldername?) \land \\
\quad fd.files = \varnothing[File] \land \\
\quad ACCESS_DELETE \in \\
\qquad ((ACL\ (folderToObject\ fd))\ requestingSubject?) \\
\qquad \bullet\ nodeName' = \{fd.location\} \lhd nodeName \land \\
\qquad FSTree'.parent = \{fd.location\} \lhd FSTree.parent \land \\
\qquad FSTree'.node = FSTree.node \setminus \{fd.location\} \land \\
\qquad ACP' = \{s : SUBJECT;\ o : OBJECT;\ m : \mathbb{P}\, ACCESSMODE \mid \\
\qquad\quad ((s, o), m) \in ACP \land o \neq folderToObject\ fd \bullet ((s, o), m)\} \land \\
\qquad history' = history \frown \\
\qquad\quad \{1 \mapsto ((((subjectComponentTypeToSUBJECT)^{\sim})requestingSubject?), \\
\qquad\quad folderToComponent\ fd, EVENT_ACCESS_DELETE)\}
\end{array}
$$

5.2.4.2 Processes and IPC

Processes can be created and terminated, they may load libraries during execution, and they communicate via IPC mechanisms (inter-process communication).

The *CreateProcess* schema includes the *FileSystem*, *Processes*, *AccessControlPolicy*, and *SystemHistory* schemas. Process creation requires provision of the name of the program (executable file) on which the process is based, the subject to be associated with the new process, and the process attempting to create a new process.

Precondition for this operation is that the program exists, that the requesting subject is allowed to execute the program and that the subject is allowed to create a new process. If the precondition is met, a new process based on the program is created and the subject associated with the new process (not to be confused with the subject requesting the

creation) is granted *full control* access for the process object. No libraries, memory areas, or input queues are associated with the newly created process. Process creation is logged in the system history.

$$
\begin{array}{l}
\textit{CreateProcess} \\
\hline
\Xi \textit{FileSystem} \\
\Delta \textit{Processes} \\
\Delta \textit{AccessControlPolicy} \\
\Delta \textit{SystemHistory} \\
\textit{program}? : \textit{PATHNAME} \\
\textit{processSubject}? : \textit{SUBJECT} \\
\textit{requestingSubject}? : \textit{SUBJECT} \\
\hline
\exists\, p : \textit{HANDLE} \bullet \\
\quad p \notin \textit{runningProcesses} \land \\
\quad \textit{ACCESS_INVOKE} \in \\
\qquad ((\textit{ACL}\ (\textit{fileToObject}\ (\textit{fileByFullName}\ \textit{program}?)))\ \textit{requestingSubject}?) \land \\
\quad \textit{ACCESS_CREATE} \in ((\textit{ACL}\ (\textit{processToObject}\ p))\ \textit{requestingSubject}?) \land \\
\quad \textit{runningProcesses}' = \textit{runningProcesses} \cup \{p\} \land \\
\quad \textit{processFile}' = \textit{processFile} \cup \{(p \mapsto (\textit{fileByFullName}\ \textit{program}?))\} \land \\
\quad \textit{processLibraries}' = \{p\} \lhd \textit{processLibraries} \land \\
\quad \textit{processMemory}' = \{p\} \lhd \textit{processMemory} \land \\
\quad \textit{processOutput}' = \{p\} \lhd \textit{processOutput} \land \\
\quad \textit{processInput}' = \{p\} \lhd \textit{processInput} \land \\
\quad \textit{processSubject}' = \textit{processSubject} \cup \{(p, \textit{processSubject}?)\} \land \\
\quad \textit{ACP}' = \textit{ACP} \cup \\
\qquad \{((\textit{processSubject}\ p, \textit{processToObject}\ p), \textit{ACCESS_GENERIC_ALL})\} \land \\
\quad \textit{history}' = \textit{history} \frown \\
\qquad \{1 \mapsto ((((\textit{subjectComponentTypeToSUBJECT})^{\sim})\textit{requestingSubject}?), \\
\qquad \textit{fileToComponent}\ (\textit{processFile}\ p), \textit{EVENT_ACCESS_INVOKE})\}
\end{array}
$$

The *TerminateProcess* schema includes the three schemas *Processes*, *AccessControlPolicy*, and *SystemHistory*. Process termination requires provision of the process to be terminated and the subject attempting to terminate the process.

Precondition for this operation is that the process exists and that the subject is allowed to terminate the process according to the current access control configuration. If the precondition is met, the process is removed from the set of running processes and all associations with libraries loaded by the process, memory areas available to the process, and input queues to which the process is attached, are removed. Process termination is not logged in the system history.

$$
\begin{array}{l}
\textit{TerminateProcess} \\
\hline
\Delta \textit{Processes} \\
\Delta \textit{AccessControlPolicy} \\
\textit{process?} : \textit{HANDLE} \\
\textit{requestingSubject?} : \textit{SUBJECT} \\
\hline
\textit{process?} \in \textit{runningProcesses} \\
\textit{ACCESS_DELETE} \in ((\textit{ACL}\ (\textit{processToObject}\ \textit{process?}))\ \textit{requestingSubject?}) \\
\textit{runningProcesses}' = \textit{runningProcesses} \setminus \{\textit{process?}\} \\
\textit{processLibraries}' = \{\textit{process?}\} \ntriangleleft \textit{processLibraries} \\
\textit{processMemory}' = \{\textit{process?}\} \ntriangleleft \textit{processMemory} \\
\textit{processInput}' = \{\textit{process?}\} \ntriangleleft \textit{processInput} \\
\textit{ACP}' = \{ s : \textit{SUBJECT};\ o : \textit{OBJECT};\ m : \mathbb{P}\,\textit{ACCESSMODE} \mid ((s, o), m) \in \textit{ACP} \land \\
\qquad o \neq \textit{processToObject}\ \textit{process?} \bullet ((s, o), m)\}
\end{array}
$$

The *LoadLibrary* schema controls process functionality extension and includes the *FileSystem*, *Processes*, *AccessControlPolicy*, and *SystemHistory* schemas. Loading of libraries requires provision of the name of the file to be created, the (existing) folder in which it shall be placed, the owner of the newly created file, and the access rights for that owner to enable the creation of an ACE of the new file.

Precondition for this operation is that there is at least one file matching the specification of the given library filename. If the full path is not specified, the operating system attempts to find a matching file in the following folders: the folder of the program for the loading process, the system folder, the main operating system folder, folders in the *SearchPath*. If more than one matching file exists, the first file in this order is loaded. Loading of libraries is logged in the system history.

$$
\begin{array}{l}
\textbf{LoadLibrary} \\
\hline
\Xi \mathit{FileSystem} \\
\Delta \mathit{Processes} \\
\Delta \mathit{LinkedLibraries} \\
\Xi \mathit{AccessControlPolicy} \\
\Delta \mathit{SystemHistory} \\
\mathit{process?} : \mathit{HANDLE} \\
\mathit{LibraryFilename?} : \mathit{PATHNAME} \\
\mathit{possibleFiles} : \mathrm{seq}\, \mathit{File} \\
\hline
(((1 \mapsto \mathit{fileByFullName}\ \mathit{LibraryFilename?}) \in \mathit{possibleFiles} \\
\quad \land \#\mathit{LibraryFilename?} > 1) \lor \\
\qquad (1 \notin \mathrm{dom}\, \mathit{possibleFiles} \land \#\mathit{LibraryFilename?} = 1)) \land \\
((2 \mapsto \mathit{fileByFullName}\ (\mathit{OSSysFolder} \frown \{1 \mapsto \mathit{last}\ \mathit{LibraryFilename?}\})) \\
\quad \in \mathit{possibleFiles} \land \\
\quad \#\mathit{LibraryFilename?} = 1) \land \\
((3 \mapsto \mathit{fileByFullName}\ (\mathit{OSFolder} \frown \{1 \mapsto \mathit{last}\ \mathit{LibraryFilename?}\})) \\
\quad \in \mathit{possibleFiles} \land \\
\quad \#\mathit{LibraryFilename?} = 1) \land \\
((4 \mapsto \mathit{fileByFullName}\ ((\mathit{front}\ (\mathit{fullNameByNode}\ (\mathit{processFile}\ \mathit{process?}).\mathit{location}))\frown \\
\quad \{1 \mapsto \mathit{last}\ \mathit{LibraryFilename?}\})) \in \mathit{possibleFiles} \land \\
\quad \#\mathit{LibraryFilename?} = 1) \land \\
(\{n : \mathbb{N};\ \mathit{pn} : \mathit{PATHNAME} \mid (n \mapsto \mathit{pn}) \in \mathit{squash}\ \mathit{SearchPath} \bullet \\
\quad (n + 4 \mapsto \mathit{fileByFullName}\ (\mathit{pn} \frown \{1 \mapsto \mathit{last}\ \mathit{LibraryFilename?}\}))\} \\
\quad \subseteq \mathit{possibleFiles} \land \\
\quad \#\mathit{LibraryFilename?} = 1) \land \\
\exists f : \mathit{File};\ l : \mathit{HANDLE} \bullet \\
\quad f = \mathit{head}\ (\mathit{squash}\ (\mathit{possibleFiles} \upharpoonright \mathrm{ran}\, \mathit{nodeFile})) \land \\
\quad \mathit{libraryFile}\ l = f \land \\
\quad \mathit{processLibraries}' = \mathit{processLibraries} \cup \{\mathit{process?} \mapsto l\} \land \\
\quad \mathit{ACCESS_INVOKE} \in ((\mathit{ACL}\ (\mathit{fileToObject}\ f))\ (\mathit{processSubject}\ \mathit{process?})) \land \\
\quad \mathit{history}' = \mathit{history}\frown \\
\qquad \{1 \mapsto (((\mathit{subjectComponentTypeToSUBJECT})^{\sim})(\mathit{processSubject}\ \mathit{process?}), \\
\qquad \mathit{fileToComponent}\ f, \mathit{EVENT_ACCESS_INVOKE})\} \\
\hline
\end{array}
$$

A process can send data to another process via an IPC mechanism. This is represented by the *InvokeIPC* schema. Data received by the target process is stored in its IPC queue for processing.

InvokeIPC

$\Delta Processes$
$FileSystem$
$\Delta SystemHistory$
$sourceProcess : HANDLE$
$targetProcess : HANDLE$
$sendData : DATA$

$$processIPCQueue' = processIPCQueue \oplus \{(targetProcess \mapsto ((processIPCQueue\ targetProcess) \frown \{1 \mapsto sendData\}))\}$$

$$history' = history \frown \{1 \mapsto (((subjectComponentTypeToSUBJECT)^{\sim})(processSubject\ sourceProcess), fileToComponent\ (processFile\ targetProcess), EVENT_ACCESS_INVOKE)\}$$

5.2.4.3 Memory

A process can store data in main memory. Memory can be allocated and deallocated. Data can be transferred between memory and memory or files. Memory areas can also be shared explicitly among processes.

Memory is allocated by a process in a part of main memory that is not in use by another process. To get access to another process's memory, *AttachMemory* has to be used. A memory area is guaranteed not to contain data when it is first used (i.e. freshness of resources applies).

The *AllocateMemory* schema includes the *Memory*, *Processes*, *AccessControlPolicy*, and *SystemHistory* schemas. Memory allocation requires provision of the process requesting a memory area. The allocated memory area is returned as *allocatedMemory*!.

There are no explicit preconditions for this operation. A so far unused memory area is associated with the requesting process. The subject executing the process is given *full control* access to the new memory area. Memory allocation is not logged in the system history.

AllocateMemory

$\Delta Memory$
$\Delta Processes$
$\Delta AccessControlPolicy$
$process? : HANDLE$
$allocatedMemory! : MemoryArea$

$$\exists\, p : MemoryArea \mid p \in memoryAreas \land \\ p.handle = allocatedMemory!.handle \land \\ allocatedMemory! \notin \operatorname{ran} processMemory \land \\ allocatedMemory!.content \in CONTENT_EMPTY \land \\ allocatedMemory!.owner = process? \\ \bullet\ memoryAreas' = memoryAreas \setminus \{p\} \cup \{allocatedMemory!\} \land \\ processMemory' = processMemory \cup \{process? \mapsto allocatedMemory!\} \land \\ ACP' = ACP \cup \{((processSubject\ process?, memoryAreaToObject\ allocatedMemory!), \\ ACCESS_GENERIC_ALL)\}$$

The *ReadMemory* schema includes the *Processes* and *SystemHistory* schemas. Reading data from a memory area requires provision of the process and of the handle of the memory area from which data is to be retrieved. Observing the data requires no special access permissions. The observed data is returned as *dataFromMemory!*.

Precondition for this operation is that the process is associated with the memory area from which data is to be read. If the precondition is met, the data is returned. Memory observation is logged in the system history.

ReadMemory

$\Xi Processes$
$\Delta SystemHistory$
$process? : HANDLE$
$memoryHandle? : HANDLE$
$dataFromMemory! : DATA$

$$\exists\, p : MemoryArea \mid p.handle = memoryHandle? \land \\ p \in processMemory(\!|\ \{process?\}\ |\!) \\ \bullet\ dataFromMemory! = p.content \land \\ history' = history ^\frown \\ \{1 \mapsto (((subjectComponentTypeToSUBJECT)^\sim)(processSubject\ process?), \\ memoryAreaToComponent\ p, EVENT_ACCESS_OBSERVE)\}$$

The *UpdateMemory* schema includes the *Processes* and *SystemHistory* schemas. Updating data in a memory area requires provision of the process, the handle of the memory area that is to be altered, and the new data to be put into the memory area. Modifying the data requires no special access permissions.

Precondition for this operation is that the process is associated with the memory area

that is to be updated. If the precondition is met, the memory page is modified to contain the new data supplied. Memory updates are logged in the system history.

UpdateMemory

$$
\begin{array}{l}
\Delta Processes \\
\Delta SystemHistory \\
process? : HANDLE \\
memoryHandle? : HANDLE \\
newMemoryData? : DATA \\
\hline
\exists\, p : MemoryArea;\ q : MemoryArea \mid p \in processMemory(\!|\ \{process?\}\ |\!) \land \\
\quad p.handle = memoryHandle? \land \\
\quad q.handle = p.handle \land \\
\quad q.owner = p.owner \land \\
\quad q.content = newMemoryData? \\
\quad \bullet\ processMemory' = processMemory \mathbin{;} \{p \mapsto q\} \land \\
\qquad history' = history ^\frown \\
\qquad\quad \{1 \mapsto (((subjectComponentTypeToSUBJECT)^\sim)(processSubject\ process?), \\
\qquad\quad memoryAreaToComponent\ p, EVENT_ACCESS_MODIFY)\}
\end{array}
$$

The *ReadFileToMemory* schema represents loading the contents of a file into an existing memory area associated with a process. It includes the *ReadFile* and *UpdateMemory* schemas. Data transfer to memory requires provision of the name of the file to be read, the memory area where it is to be stored, and a process associated with that memory area.

Precondition for this operation is that the file exists and that the requesting subject associated with the process has *observe* access to the file. If the precondition is met, the file's content is transferred to the memory area associated with the process. Data transfer to memory is logged in the system history.

$$
\begin{array}{l}
ReadFileToMemory == ReadFile[newMemoryData?/filecontent!] \land \\
[UpdateMemory;\ requestingSubject? : SUBJECT \mid \\
processSubject\ process? = requestingSubject?]
\end{array}
$$

The *UpdateFileFromMemory* schema represents exporting the contents of a memory area to an existing file. It includes the *UpdateFile* and *ReadMemory* schemas. Data transfer from memory requires provision of the memory area to be read from, a process associated with it, and a file where the data is to be stored.

Precondition for this operation is that the file exists and that the requesting subject associated with the process has *modify* access to the file. If the precondition is met, the memory area's content is transferred to the file. Data transfer from memory is logged in the system history.

$$
\begin{array}{l}
UpdateFileFromMemory == UpdateFile[dataFromMemory!/filecontent?] \land \\
[ReadMemory;\ requestingSubject? : SUBJECT \mid \\
processSubject\ process? = requestingSubject?]
\end{array}
$$

The *CopyMemoryContent* schema represents transfer from one memory area to another, and includes the *ReadMemory* and *UpdateMemory* schemas. Data transfer in memory requires provision of the memory area to be read from and the memory area where the data is to be stored.

Precondition for this operation is that the memory areas are both associated with the process attempting the data transfer. If the precondition is met, the source memory area's content is transferred to the target memory area. Data transfer in memory is logged in the system history.

$$
\begin{aligned}
&\mathit{CopyMemoryContent} == \\
&\quad \mathit{ReadMemory} \land \\
&\quad \mathit{UpdateMemory}[\mathit{dataFromMemory}!/\mathit{newMemoryData}?, \\
&\qquad \mathit{targetMemoryHandle}?/\mathit{memoryHandle}?]
\end{aligned}
$$

The *AttachMemory* schema includes the *Processes* and *AccessControlPolicy* schemas. Attaching additional memory areas to a process requires provision of the process and the handle of the memory area to attach to.

Precondition for this operation is that the memory area referenced by its handle exists and that the access permissions allow *observation* and *modification* of the referenced memory area by the requesting subject associated with the process. If the precondition is met, the memory area is associated with the process so that access by, e.g., *ReadMemory* is made possible. Attachment of memory areas to processes is not logged in the system history, but accesses to attached memory areas are.

AttachMemory

$$
\begin{aligned}
&\Delta \mathit{Processes} \\
&\Xi \mathit{AccessControlPolicy} \\
&\mathit{process}? : \mathit{HANDLE} \\
&\mathit{memoryHandle}? : \mathit{HANDLE} \\
\hline
&\exists\, m : \mathit{MemoryArea} \mid m \in \operatorname{ran} \mathit{processMemory} \land \\
&\quad m.\mathit{handle} = \mathit{memoryHandle}? \land \\
&\quad (\{\mathit{ACCESS_OBSERVE}, \mathit{ACCESS_MODIFY}\} \\
&\qquad \subseteq ((\mathit{ACL}\ (\mathit{memoryAreaToObject}\ m))\ (\mathit{processSubject}\ \mathit{process}?))) \\
&\quad \bullet\ \mathit{processMemory}' = \mathit{processMemory} \cup \{\mathit{process}? \mapsto m\}
\end{aligned}
$$

The *DetachMemory* schema includes the *Processes* schema. Detaching from a memory area does not require special permissions by the subject associated with the processes currently associated with the memory area in question.

Precondition for this operation is that the referenced memory area exists and is associated with the process requesting the disassociation. If the precondition is met, the association between the process and the memory area is removed. If no other processes are associated with that memory area, its contents are no longer available to any process in the system. Reallocation will return a memory area with empty content, and attachment is only possible to memory areas associated with processes. Detaching from a memory area is not logged in the system history.

$$
\begin{array}{l}
\textit{DetachMemory} \\
\hline
\Delta \textit{Processes} \\
\textit{process}? : \textit{HANDLE} \\
\textit{memoryHandle}? : \textit{HANDLE} \\
\hline
\exists\, p : \textit{MemoryArea} \mid p.\textit{handle} = \textit{memoryHandle}? \land \\
\quad p \in \textit{processMemory} (\!| \{\textit{process}?\} |\!) \\
\quad \bullet\ \textit{processMemory}' = \textit{processMemory} \setminus \{\textit{process}? \mapsto p\}
\end{array}
$$

The *DeallocateMemory* schema includes the three schemas *Processes*, *AccessControlPolicy*, and *SystemHistory*. Deallocating a memory area requires provision of the handle of the memory area to be deallocated and the process attempting to deallocate.

Preconditions for this operation are that the memory area exists and that it is associated with the process attempting to deallocate. In addition, the subject executing the process needs *delete* access to the memory area in question. If the precondition is met, all associations between processes and the memory area are removed. The memory area's contents are no longer available to any process in the system. Reallocation will return a memory area with empty content, and attachment is only possible to memory areas associated with processes. Deallocating a memory area is not logged in the system history.

$$
\begin{array}{l}
\textit{DeallocateMemory} \\
\hline
\Delta \textit{Processes} \\
\Delta \textit{AccessControlPolicy} \\
\textit{process}? : \textit{HANDLE} \\
\textit{memoryHandle}? : \textit{HANDLE} \\
\hline
\exists\, p : \textit{MemoryArea} \mid p.\textit{handle} = \textit{memoryHandle}? \land \\
\quad p \in \textit{processMemory} (\!| \{\textit{process}?\} |\!) \land \\
\quad \textit{ACCESS_DELETE} \in ((\textit{ACL}\ (\textit{memoryAreaToObject}\ p))\ (\textit{processSubject}\ \textit{process}?)) \\
\quad \bullet\ \textit{processMemory}' = \textit{processMemory} \rhd \{p\} \land \\
\qquad \textit{ACP}' = \{s : \textit{SUBJECT};\ o : \textit{OBJECT};\ m : \mathbb{P}\, \textit{ACCESSMODE} \mid \\
\qquad\quad ((s, o), m) \in \textit{ACP} \land o \neq \textit{memoryAreaToObject}\ p \bullet ((s, o), m)\}
\end{array}
$$

5.2.4.4 Access control

We treat all access rights as fixed in this version of the model for two reasons. First, we need to keep the process model simple and compatible to a large number of operating systems. It is not given that an ordinary user (and hence, an attacker) can modify access rights. Second, computing the transitive closure of non-monotonic access control configurations has been shown to be $NP - complete$ (cf. [HRU76]). This might impair our ability to reason about states in the model.

Extension of our model by adding schemas for explicit changes in the access control

configuration is nevertheless possible and left for future work.

5.2.4.5 User and user interface

Processes can observe and modify the output devices of the user interface, provided they have the necessary access permissions. Precondition for this operation is that the output device exists (referenced by its handle).

$$
\begin{array}{l}
\textit{ReadUIOutput} \\
\hline
\Xi \textit{UIOutputDevices} \\
\Xi \textit{AccessControlPolicy} \\
\Delta \textit{SystemHistory} \\
\textit{requestingSubject}? : \textit{SUBJECT} \\
\textit{outputDevice} : \textit{HANDLE} \\
\textit{UIData} : \textit{DATA} \\
\hline
\textit{UIData} = \textit{displayedData}\ \textit{outputDevice} \wedge \\
\textit{ACCESS_OBSERVE} \in ((\textit{ACL}\ (\textit{UIOutputDeviceToObject}\ \textit{outputDevice}))\ \textit{requestingSubject}?) \\
\textit{history}' = \textit{history} ^\frown \\
\quad \{1 \mapsto (((\textit{subjectComponentTypeToSUBJECT})^\sim)\textit{requestingSubject}?, \\
\quad\quad \textit{uiOutputDeviceToComponent}\ \textit{outputDevice}, \textit{EVENT_ACCESS_OBSERVE})\}
\end{array}
$$

$$
\begin{array}{l}
\textit{WriteUIOutput} \\
\hline
\Delta \textit{UIOutputDevices} \\
\Xi \textit{AccessControlPolicy} \\
\Delta \textit{SystemHistory} \\
\textit{requestingSubject}? : \textit{SUBJECT} \\
\textit{outputDevice} : \textit{HANDLE} \\
\textit{UIData} : \textit{DATA} \\
\hline
\textit{displayedData}' = \textit{displayedData} \oplus \{(\textit{outputDevice}, \textit{UIData})\} \wedge \\
\textit{ACCESS_MODIFY} \in ((\textit{ACL}\ (\textit{UIOutputDeviceToObject}\ \textit{outputDevice}))\ \textit{requestingSubject}?) \\
\textit{history}' = \textit{history} ^\frown \\
\quad \{1 \mapsto (((\textit{subjectComponentTypeToSUBJECT})^\sim)\textit{requestingSubject}?, \\
\quad\quad \textit{uiOutputDeviceToComponent}\ \textit{outputDevice}, \textit{EVENT_ACCESS_MODIFY})\}
\end{array}
$$

Capturing and synthesizing user input is performed by the two schemas *ReadUIInput* and *WriteUIInput*. Precondition for this operation is that the output device exists (referenced by its handle). An input event is removed from the input queue after it has been read.

$$
\begin{array}{l}
\textit{ReadUIInput} \\
\hline
\Delta \mathit{UIInputDevices} \\
\Xi \mathit{AccessControlPolicy} \\
\Delta \mathit{SystemHistory} \\
\mathit{requestingSubject?} : \mathit{SUBJECT} \\
\mathit{inputDevice} : \mathit{HANDLE} \\
\mathit{UIData} : \mathit{DATA} \\
\hline
\mathit{UIData} = \mathit{head}\ (\mathit{inputEvents}\ \mathit{inputDevice}) \wedge \\
\mathit{inputEvents}'\ \mathit{inputDevice} = \mathit{tail}\ (\mathit{inputEvents}\ \mathit{inputDevice}) \wedge \\
\mathit{ACCESS_OBSERVE} \in ((\mathit{ACL}\ (\mathit{UIInputDeviceToObject}\ \mathit{inputDevice}))\ \mathit{requestingSubject?}) \\
\mathit{history}' = \mathit{history} \frown \\
\quad \{1 \mapsto (((\mathit{subjectComponentTypeToSUBJECT})^{\sim})\mathit{requestingSubject?}, \\
\qquad \mathit{uiInputDeviceToComponent}\ \mathit{inputDevice}, \mathit{EVENT_ACCESS_OBSERVE})\}
\end{array}
$$

$$
\begin{array}{l}
\textit{WriteUIInput} \\
\hline
\Delta \mathit{UIInputDevices} \\
\Xi \mathit{AccessControlPolicy} \\
\Delta \mathit{SystemHistory} \\
\mathit{requestingSubject?} : \mathit{SUBJECT} \\
\mathit{inputDevice} : \mathit{HANDLE} \\
\mathit{UIData} : \mathit{DATA} \\
\hline
\mathit{inputEvents}' = \mathit{inputEvents} \oplus \{(\mathit{inputDevice}, (\mathit{inputEvents}\ \mathit{inputDevice}) \frown \\
\quad \{1 \mapsto \mathit{UIData}\})\} \wedge \\
\mathit{ACCESS_MODIFY} \in ((\mathit{ACL}\ (\mathit{UIInputDeviceToObject}\ \mathit{inputDevice}))\ \mathit{requestingSubject?}) \\
\mathit{history}' = \mathit{history} \frown \\
\quad \{1 \mapsto (((\mathit{subjectComponentTypeToSUBJECT})^{\sim})\mathit{requestingSubject?}, \\
\qquad \mathit{uiInputDeviceToComponent}\ \mathit{inputDevice}, \mathit{EVENT_ACCESS_MODIFY})\}
\end{array}
$$

5.3 Limitations of the model

Of course, one often wishes to have an all-encompassing model to explain each and every facet of a problem. The accompanying danger is to overspecify and to recreate a whole operating system for real systems without the necessary abstraction needed to deal with the architectural security of a program. It is easy to misuse Z as an arcane assembly language for that purpose.

You do not have a product if you do not define what you are not going to do. In that sense, some areas of our model of a generic operating system are left to future work.

This encompasses, but is not limited to the following:

- Tamper-proof storage can be specified in an architectural description, but is cur-

rently not handled in the operating system model. It is possible to use a common *data* component and explicitly specify appropriate *access rights* instead of relying on the implicit access restriction of a tamper-proof storage component.

- *ACCESS_CREATE* connectors are not mapped from an architectural description to operating system entities. The *CreateFile* schema does not take into account the owner of a new file depending on the container component of the corresponding architectural description.
- Access control configuration data and logging configuration data are not present as objects in the operating system model.
- References to files and folders can be specified in an architectural description, but are currently not handled in the operating system model.
- Processes launched automatically, e.g., by a mechanism of the operating system, are not supported.
- Inter-process communication could be handled differently depending on whether or not it is based on an authenticated data transfer connector in the corresponding architectural description.
- Monitoring of operations and alerting the local human user is currently not expressible in the operating system model.

Model definitions index

!, 109
Δ*schema name*, 108
Ξ*schema name*, 109
?, 109

ACCESS_APPEND (constant), 91
ACCESS_CREATE (constant), 91
ACCESS_DELETE (constant), 91
ACCESS_GENERIC_ALL (constant), 91
ACCESS_INVOKE (constant), 91
ACCESS_MODIFY (constant), 91
ACCESS_OBSERVE (constant), 91
accessConnectorTypes (set), 91
AccessControlPolicy (schema), 98
accessLogged (schema variable), 167
ACCESSMODE (given set), 91
accessMonitored (schema variable), 167
ACL (function), 98
ACM (given set), 91
ACM (relation), 91
AddExecutableFileInFolder, 140
AddExecutableFileInFolder (schema), 139
adversarialProcess (schema variable), 147
AllComponentsCovered (schema), 108
allocatedMemory! (schema variable), 119
AllocateMemory (schema), 119
appended (relation), 90, 112
AppendFile (schema), 112
areaByHandle (function), 96
AttachMemory (schema), 122
attachment (function), 100
AttackAddStoredCodeModule (schema), 140
AttackAddStoredCodeModulePrologue (schema), 138
Attacker (schema), 162
AttackerCapabilities (relation), 161
AttackingProcess (schema variable), 128, 132, 134, 138, 141, 144, 151, 155
AttackingProcess (schema), 153
AttackInitiateCommunicationAndSendData (schema), 146
AttackInitiateCommunicationAndSendDataPrologue (schema), 144
AttackModifyCodeInMemory (schema), 135
AttackModifyCodeInMemoryPrologue (schema), 134
AttackModifyReferenceToStoredCodeModule (schema), 143
AttackModifyReferenceToStoredCodeModulePrologue (schema), 141
AttackModifyReferenceToStoredParameters (schema), 152
AttackModifyReferenceToStoredParametersPrologue (schema), 151
AttackModifyStoredCodeModule (schema), 137
AttackModifyStoredDataComponent (schema), 131
AttackModifyStoredDataComponentForDecisions (schema), 157
AttackModifyStoredDataComponentPrologue (schema), 128
AttackModifyStoredParameters (schema), 150
AttackModifyUserInterfaceObject (schema), 156
AttackModifyUserInterfaceObjectPrologue (schema), 155
AttackObserveStoredDataComponent (schema), 133
AttackObserveStoredDataComponentPrologue (schema), 132
AttackRespondCommunicationAndSendData (schema), 148
AttackSimulateUserInput (schema), 154
AttackSimulateUserInputPrologue (schema), 153
AttCapInit (given set), 160
AttCapTime (given set), 160

AttCapUser (given set), 161
AttCapVariation (given set), 160
AttemptAttackAddStoredCodeModule (schema), 164, 167
AttemptAttackInitiateCommunicationAndSendData (schema), 164, 167
AttemptAttackModifyCodeInMemory (schema), 163, 167
AttemptAttackModifyReferenceToStoredCodeModule (schema), 164, 167
AttemptAttackModifyReferenceToStoredParameters (schema), 165, 167
AttemptAttackModifyStoredCodeModule (schema), 163, 167
AttemptAttackModifyStoredDataComponent (schema), 162, 167
AttemptAttackModifyStoredDataComponentForDecisions (schema), 166, 167
AttemptAttackModifyStoredParameters (schema), 165, 167
AttemptAttackModifyUserInterfaceObject (schema), 166, 167
AttemptAttackObserveStoredDataComponent (schema), 163, 167
AttemptAttackRespondCommunicationAndSendData (schema), 164, 167
AttemptAttacks (schema), 167
AttemptAttackSimulateUserInput (schema), 165, 167

CAP (function), 98
capabilities (schema variable), 162
child (schema variable), 93
COMPNAME (given set), 89
Component (schema), 100
components (function), 100
componentType (schema variable), 100
COMPTYPE (given set), 88
condag (set), 93
Configuration (schema), 100
connectionType (schema variable), 100
Connector (schema), 100
connectors (function), 100
CONNNAME (given set), 89
CONNTYPE (given set), 88
ContainerComponent (schema variable), 138, 139
content (schema variable), 93, 96
CONTENT_ATTACKERS_CHOICE (constant), 90
CONTENT_EMPTY (constant), 90
CONTENT_EXECUTABLE (constant), 90
CopyFile (schema), 113
CopyMemoryContent (schema), 122
correspondingLibraries (relation), 103
correspondingMemoryAreas (relation), 104
correspondingUIInput (relation), 105
correspondingUIOutput (relation), 105
CreateFile (schema), 109
CreateFileAsAttacker (schema), 130
CreateFolder (schema), 114
CreateProcess (schema), 116
currentAccessMode (function), 130

dag (set), 92
DATA (given set), 90
DataComponent (schema variable), 129, 130, 132, 136, 137, 149, 157
dataFromMemory (schema variable), 121
dataFromMemory! (schema variable), 120, 122
DeallocateMemory (schema), 123
DeleteFile (schema), 113
DeleteFolder (schema), 115
DeleteStoredDataComponent (schema), 129
DesiredSecurityRequirements (schema variable), 172, 181
DetachMemory (schema), 122
DetermineResistanceClass (schema), 172, 181
digraph (set), 92
displayedData (function), 96

EmptyState (schema), 101
EvaluateAttackedSystem (schema), 172
EvaluateCodeIntegrity (schema), 170
EvaluateDataConfigentiality (schema), 169
EvaluateDataIntegrity (schema), 168
EvaluateSystemState (schema), 172
EVENT_ACCESS_APPEND (constant), 92
EVENT_ACCESS_CREATE (constant), 92
EVENT_ACCESS_DELETE (constant), 92
EVENT_ACCESS_INVOKE (constant), 92
EVENT_ACCESS_MODIFY (constant), 92

EVENT_ACCESS_OBSERVE (constant), 92
EVENT_ACCESS_RENAME (constant), 92
EVENTACTION (given set), 92

File (schema), 93
fileByFullName (function), 95
filecontent! (schema variable), 110, 121, 133
filecontent? (schema variable), 111, 112, 121, 131, 137
filename (schema variable), 93
filename? (schema variable), 110–113
files (schema variable), 94
FileSystem (schema), 95
fileToComponent (function), 95
fileToObject (function), 98
FindContainer (schema), 138
FindExecutableFileOfAdversarialProcess-Modules (schema), 147
FindExecutableFileOfVictimProcessModules (schema), 144
FindIPCComponentOfAdversarialProcess (schema), 148
FindIPCComponentOfVictimProcess (schema), 145
FindReference (schema), 141
FindStoredDataComponent (schema), 129
FindVictimMemoryArea (schema), 135
FindVictimProcess (schema), 135
FindVictimUIInputQueue (schema), 153
FindVictimUIOutputDevice (schema), 155
Folder (schema), 94
folderByFullName (function), 95
foldername (schema variable), 94
foldername? (schema variable), 115
folderToComponent (function), 95
folderToObject (function), 98
FSNODE (given set), 92
fsnodeToObject (function), 98
fullNameByNode (function), 95

HANDLE (given set), 92
handle (schema variable), 96
HighestSatisfiableSecurityRequirements (schema variable), 168–170
history (schema variable), 99

InitialAccessControlPolicy (schema), 107
InitialACE (function), 107
InitialConfigurationACE (schema), 107
InitialFileSystem (schema), 101
InitialMemory (schema), 102
InitialProcesses (schema), 106
InitialStateCorrespondence (schema), 108
InitialUI (schema), 103
inputDevice (schema variable), 124, 125
inputEvents (function), 96
InvokeIPC (schema), 118
IPCResponse (function), 90
IsContainerUsedInExecutionReference (schema), 139
IsReferenceToCodeModule (schema), 142
IsReferenceToParameters (schema), 151
IsStoredDataComponentExecutable (schema), 136
IsStoredDataComponentUsedInDecision (schema), 157
IsVictimMemoryAreaExecutable (schema), 135

leaf (schema variable), 93
libraryFile (function), 97
LibraryFilename? (schema variable), 117
LinkedLibraries (schema), 97
loadedLibraries (schema variable), 97
LoadLibrary (schema), 117
location (schema variable), 93, 94
LoggedComponents (schema), 167

MappableComponentTypes (set), 88
Memory (schema), 96
MemoryArea (schema), 96
memoryAreas (schema variable), 96
memoryAreaToComponent (function), 96
memoryAreaToObject (function), 98
memoryHandle? (schema variable), 120–123
ModifyReference (schema), 142
MonitoredComponents (schema), 167
MoveFile (schema), 114

NAME (given set), 89
newFileAccessMode? (schema variable), 109
newFilename? (schema variable), 109, 112, 113

newFolderAccessMode? (schema variable), 114
newFoldername? (schema variable), 114
newFolderOwner? (schema variable), 114
newMemoryData? (schema variable), 121, 122, 135
node (schema variable), 93
nodeByFullName (function), 95
nodeData (function), 95
nodeFile (function), 95
nodeFolder (function), 95
nodeName (function), 95

OBJECT (given set), 90
oldFilename? (schema variable), 112
OSFolder (constant), 94
OSSysFolder (constant), 94
outputDevice (schema variable), 124
OWNER (given set), 100
owner (schema variable), 96, 100

parent (schema variable), 93
parentfolder? (schema variable), 114
PATHNAME (sequence), 92
PORT (given set), 89
PORTINST (given set), 89
PORTINST (relation), 89
ports (schema variable), 100
possibleFiles? (schema variable), 117
process? (schema variable), 116, 117, 119–123
Processes (schema), 97
processFile (function), 97
processInput (relation), 97
processIPCQueue (function), 97
processIPCResponse (function), 97
processLibraries (relation), 97
ProcessLibraryCorrespondence (schema), 103
processMemory (relation), 97
ProcessMemoryCorrespondence (schema), 104
processOutput (relation), 97
processSubject (function), 97
processSubject? (schema variable), 116
processToObject (function), 98
ProcessUIInputCorrespondence (schema), 105
ProcessUIOutputCorrespondence (schema), 105
ProductSecurityStatus (schema variable), 172
program? (schema variable), 116

ReadFile (schema), 110
ReadFileToMemory (schema), 121
ReadMemory (schema), 120
ReadUIInput (schema), 124
ReadUIOutput (schema), 124
ReferenceComponent (schema variable), 141, 142, 151
RenameFile (schema), 112
ReplaceStoredDataComponent (schema), 131
ReplaceStoredDataComponentPrologue (schema), 130
requestingSubject? (schema variable), 109–116, 121, 124, 125, 130, 131, 133, 140, 143, 152, 154, 156
ResistanceClass (schema variable), 172
RetrieveStoredDataComponent (schema), 132
ROLE (given set), 89
ROLEINST (given set), 89
ROLEINST (relation), 89
roles (schema variable), 100
root (schema variable), 93
runningProcesses (schema variable), 97

SatisfyingSecurityRequirements (relation), 171
SearchPath (given set), 94, 117
SecReqCodeInt (given set), 159
SecReqCodeInt (relation), 168
SecReqDataConf (given set), 159
SecReqDataConf (relation), 168
SecReqDataInt (given set), 159
SecReqDataInt (relation), 168
SecurityRequirements (relation), 159
SecurityRequirementsRestriction (schema), 168
SECURITYSTATUS (given set), 172
sendData (schema variable), 118
sourceProcess (schema variable), 118
StoredDataComponentContainsParameters (schema), 149

subfolders (function), 95
SUBJECT (given set), 90
SUBJECT_ADVERSARY (constant), 90
SUBJECT_OS (constant), 90
SUBJECT_UNSPECIFIED (constant), 90
SUBJECT_VICTIM (constant), 90
subjectComponentTypes (set), 90
subjectComponentTypeToSUBJECT (function), 90, 91
subjectToObject (function), 98
SufficientCapabilities (relation), 162
SYSTEMEVENT (given set), 92
SystemHistory (schema), 99

t: tree[N] (schema variable), 93
TargetComponent (schema variable), 141
targetMemoryHandle? (schema variable), 122
targetProcess (schema variable), 118
TerminateProcess (schema), 116
Tree (schema), 93
tree (set), 93
typesAndComponents (function), 100

UIData (schema variable), 124, 125
uiInputDeviceComponent (function), 96
UIInputDevices (schema), 96
UIInputDeviceToObject (function), 98
uiOutputDeviceComponent (function), 96
UIOutputDevices (schema), 96
UIOutputDeviceToObject (function), 98
UpdateFile (schema), 111
UpdateFileFromMemory (schema), 121
UpdateMemory (schema), 121
UpdateStoredDataComponent (schema), 129
UserInterface (schema), 97

VictimMemoryArea (schema variable), 135
VictimProcess (schema variable), 135
victimProcess (schema variable), 144
victimUIInputQueue (schema variable), 153
victimUIOutputDevice (schema variable), 155

WriteUIInput (schema), 125
WriteUIOutput (schema), 124

Chapter 6

Formal model of generic attacks

Chapter summary: In this chapter we formally specify the repository of generic attack methods developed in chapter 3. Attacks are formulated in terms of our model of a generic computer system, presented in the previous chapters. The attack repository is complemented by a formal specification of the metrics for security requirements and attacker capabilities. We show how an evaluator can verify whether a given architectural description complies with a desired resistance class for a product.

6.1 Repository of generic malware attacks

In chapter 3 we identified 13 generic attack methods available to local malware (cf. table 3.2). They are now presented together with a formal specification in Z.

Local malware attacks can be categorised as one of the following:

Direct violation of integrity, confidentiality of stored data: Modify stored data component, Retrieve contents of stored data component

Violation of integrity of executed code: Modify code in memory, Modify stored code module, Add stored code module, Modify reference to stored code module

Violation of integrity of parameters: Initiate communication with component and send data, Respond to component's communication request and send data, Modify stored data item containing parameters, Modify reference to stored data item containing parameters, Simulate user input

Influence on user: Modify user interface object, Modify stored data item examined by users as basis for decisions

The reader is advised that there is an index of all types and schema definitions on page 127. It can be used when browsing the attack specifications to easily retrieve the location of used types and schemas.

6.1.1 Direct violation of integrity, confidentiality of stored data

Attacks in this class comprise methods by which an attacker modifies, deletes, or replaces stored data components or retrieves data from stored data components, using the procedures offered by the operating system, and doing so without cooperation of or dependence on the process under attack.

6.1.1.1 Modify stored data component

Attack description The attacker tries to modify, delete, or replace a stored data component. The attacker has to be able to identify a data component of the victim. Success of the attack depends on the data component existing and its modification by the attacking process not being restricted by access rights. Only the interfaces of the operating system intended for modification of data components are used.

Preconditions

1. The attacker must be able to find a victim's stored data component (any).
2. The data component must exist.
3. The access control configuration must permit modification.

Postconditions

1. The data component is modified, i.e., it contains data of the attacker's choice.

Architectural structures
Attacking process, data component, access control configuration.

Metrics
M_5 Percentage of access control instrumentation (table 3.7 on page 63) focuses on protection of all entry points to a component, M_6 Conformity of access permissions (table 3.8 on page 63) focuses on identical permissions to ease correct administration.

Specification in Z
All attack methods – modifying, deleting, replacing – are performed by an adversarial process. This process must be among the running processes; otherwise, the adversary would have no means to attack with. We remind the reader that we focus on local malware attacks.

$$
\begin{array}{l}
\textit{AttackModifyStoredDataComponent}_{\textit{Prologue}} \\
\hline
\Xi \textit{Processes} \\
\textit{AttackingProcess} : \textit{HANDLE} \\
\hline
\textit{AttackingProcess} \in \textit{runningProcesses}
\end{array}
$$

An attacked stored data component must be identifiable as belonging to the program the adversary wants to attack. This is achieved by checking the *owner* property of the component. A stored data component is associated with a file in the file system.

$$
\begin{array}{l}
\textit{FindStoredDataComponent} \\
\hline
\Xi \textit{Configuration} \\
\textit{FileSystem} \\
\textit{DataComponent} : \textit{COMPNAME} \\
\hline
(\textit{components DataComponent}).\textit{componentType} = \textit{CTDATA} \\
(\exists\, \textit{connname} : \textit{CONNNAME} \bullet \\
\exists\, \textit{ocn} : \textit{COMPNAME} \bullet \exists\, \textit{ocp} : \textit{PORT} \bullet \exists\, \textit{cp} : \textit{PORT} \bullet \\
\quad (\textit{connectors connname}).\textit{connectionType} = \textit{CNOWNER} \land \\
\quad (\textit{components ocn}).\textit{componentType} = \textit{CTSUBJECTVICTIM} \land \\
\quad ((\textit{connname}, \textit{ROSUBJECT}), (\textit{ocn}, \textit{ocp})) \in \textit{attachment} \land \\
\quad ((\textit{connname}, \textit{ROOBJECT}), (\textit{DataComponent}, \textit{cp})) \in \textit{attachment}) \\
\exists\, f : \textit{File} \bullet \textit{fileToComponent}\ f = \textit{DataComponent} \land \\
\quad f.\textit{location} \in \textit{FSTree}.\textit{node}
\end{array}
$$

If a component of interest can be found, the *UpdateFile* operation can be applied to the file, modifying the content of the file based on the attacker's choice. Whether this succeeds or triggers an alarm, depends on the predicate part of *UpdateFile* (cf. page 111).

$$
\begin{array}{l}
\textit{UpdateStoredDataComponent} \\
\hline
\Xi \textit{Configuration} \\
\textit{UpdateFile} \\
\textit{DataComponent} : \textit{COMPNAME} \\
\hline
\textit{filecontent?} \in \textit{CONTENT_ATTACKERS_CHOICE} \\
\exists\, f : \textit{File} \bullet \\
\quad (\textit{fileByFullName filename?} = f \land \\
\quad \textit{fileToComponent}\ f = \textit{DataComponent})
\end{array}
$$

Alternatively, if a component of interest can be found, the *DeleteFile* operation can be applied to it, removing the file and its content from the file system. Whether this succeeds or triggers an alarm, depends on the predicate part of *DeleteFile* (cf. page 113).

$$
\begin{array}{l}
\textit{DeleteStoredDataComponent} \\
\hline
\Xi \textit{Configuration} \\
\textit{DeleteFile} \\
\textit{DataComponent} : \textit{COMPNAME} \\
\hline
\exists f : \textit{File} \bullet \\
\quad (\textit{fileByFullName}\ \textit{filename}? = f \land \\
\quad \textit{fileToComponent}\ f = \textit{DataComponent})
\end{array}
$$

If a component of interest can be found, it can also be replaced by deleting the file and creating a new one with the same name, but content based on the attacker's choice. First, the current access rights for the file are captured so that they can be restored for the file to be created.

$$
\begin{array}{l}
\textit{ReplaceStoredDataComponent}_{\textit{Prologue}} \\
\hline
\Xi \textit{Configuration} \\
\textit{AccessControlPolicy} \\
\textit{DataComponent} : \textit{COMPNAME} \\
\textit{currentAccessMode} : \textit{SUBJECT} \rightarrow \mathbb{P}\ \textit{ACCESSMODE} \\
\textit{requestingSubject}? : \textit{SUBJECT} \\
\hline
\exists f : \textit{File} \bullet \\
\quad \textit{fileToComponent}\ f = \textit{DataComponent} \land \\
\quad \textit{currentAccessMode} = (\textit{ACL}\ (\textit{fileToObject}\ f))
\end{array}
$$

A new file is created by the adversarial process and access rights are set according to the saved ones.

$$
\begin{array}{l}
\textit{CreateFileAsAttacker} \\
\hline
\Xi \textit{Configuration} \\
\textit{CreateFile}[\textit{filename}?/\textit{newFilename}?] \\
\textit{currentAccessMode} : \textit{SUBJECT} \rightarrow \mathbb{P}\ \textit{ACCESSMODE} \\
\textit{requestingSubject}? : \textit{SUBJECT} \\
\hline
\textit{newFileAccessMode}? = \textit{currentAccessMode}
\end{array}
$$

Replacing a stored data component can then be expressed as the composition of deletion, creation, and update.

$$
\begin{aligned}
&\mathit{ReplaceStoredDataComponent} == \\
&\quad \mathit{ReplaceStoredDataComponent}_{\mathit{Prologue}} \land (\mathit{DeleteStoredDataComponent} \fatsemi \\
&\quad \mathit{CreateFileAsAttacker}) \fatsemi \\
&\quad [\mathit{Configuration};\ \mathit{UpdateFile} \mid \\
&\qquad \mathit{filecontent?} \in \mathit{CONTENT_ATTACKERS_CHOICE}]
\end{aligned}
$$

An attack on the integrity of a stored data component consists of finding a suitable component to attack, composed with the disjunction of update, deletion, and replacement.

$$
\begin{aligned}
&\mathit{AttackModifyStoredDataComponent} == \\
&\quad [\mathit{AttackModifyStoredDataComponent}_{\mathit{Prologue}} \land \mathit{FindStoredDataComponent} \land \\
&\quad (\mathit{UpdateStoredDataComponent} \lor \mathit{DeleteStoredDataComponent} \\
&\qquad \lor \mathit{ReplaceStoredDataComponent}) \\
&\quad \mid \mathit{requestingSubject?} = \mathit{processSubject}\ \mathit{AttackingProcess}]
\end{aligned}
$$

Necessary attacker capabilities

Attack initiation capability: Attacker needs to be able to launch attack at own discretion
Available time to attack: Attacker needs to be able to carry out attack at least once per user session
Attack variation: Attacker needs not to be able to customise attack
Influence on user: Attacker needs not to be able to influence the user

6.1.1.2 Retrieve contents of stored data component

Attack description The attacker tries to retrieve the content of a stored data component. The attacker has to be able to identify a data component of the victim. Success of the attack depends on the data component existing and its observation by the attacking process not being restricted by access rights. Only the interfaces of the operating system intended for observation of data components are used.

Preconditions

1. The attacker must be able to find a victim's stored data component (any).
2. The data component must exist.
3. The access control configuration must permit observation.

Postconditions

1. The content of the data component is known to the attacking process.

Architectural structures
Attacking process, data component, access control configuration.

Metrics
M_5 Percentage of access control instrumentation (table 3.7 on page 63) focuses on protection of all entry points to a component, M_6 Conformity of access permissions (table 3.8 on page 63) focuses on identical permissions to ease correct administration.

Specification in Z
Observation is performed by an adversarial process. This process must be among the running processes.

$$
\begin{array}{|l}
\hline
\multicolumn{1}{l}{\textit{AttackObserveStoredDataComponent}_{\textit{Prologue}}} \\
\Delta \textit{Processes} \\
\textit{AttackingProcess} : \textit{HANDLE} \\
\hline
\textit{AttackingProcess} \in \textit{runningProcesses} \\
\hline
\end{array}
$$

If a component of interest can be found, the *ReadFile* operation can be applied to it, retrieving the content of the file based on the attacker's choice. Whether this succeeds or triggers an alarm, depends on the predicate part of *ReadFile* (cf. page 110).

$$
\begin{array}{l}
\textbf{RetrieveStoredDataComponent} \\
\hline
\Xi \mathit{Configuration} \\
\mathit{ReadFile} \\
\mathit{DataComponent} : \mathit{COMPNAME} \\
\hline
\exists f : \mathit{File} \bullet \\
\quad (\mathit{fileToComponent}\ f = \mathit{DataComponent} \land \\
\quad \mathit{fileByFullName}\ \mathit{filename}? = f)
\end{array}
$$

An attack on the confidentiality of a stored data component consists of finding a suitable component to attack, followed by a read operation.

$$
\begin{array}{l}
\mathit{AttackObserveStoredDataComponent} == \\
\quad [\mathit{AttackObserveStoredDataComponent}_{\mathit{Prologue}} \land \mathit{FindStoredDataComponent} \land \\
\quad \mathit{RetrieveStoredDataComponent} \\
\quad \mid \mathit{requestingSubject}? = \mathit{processSubject}\ \mathit{AttackingProcess} \land \\
\quad \exists\, \mathit{ma} : \mathit{MemoryArea} \bullet \mathit{ma.content} = \mathit{filecontent}! \land \\
\qquad (\mathit{AttackingProcess}, \mathit{ma}) \in \mathit{processMemory}']
\end{array}
$$

Necessary attacker capabilities
Attack initiation capability: Attacker needs to be able to launch attack at own discretion
Available time to attack: Attacker needs to be able to carry out attack at least once per user session
Attack variation: Attacker needs not to be able to customise attack
Influence on user: Attacker needs not to be able to influence the user

6.1.2 Violation of integrity of executed code

The attacker tries to modify, delete, or replace a component containing executable code. The attacker has to be able to identify a code component of the victim process and to find it in the data store. Success of the attack depends on the code component existing and its modification by the attacking process not being restricted by access rights.

6.1.2.1 Modify code in memory

Attack description Like 6.1.1.1, but the attacker's process must know the attacked process und must have access to the attacked process's memory according to the access permissions.

Preconditions

1. The attacker must be able to find a victim's code component in memory (any).
2. The code component must exist.
3. The access control configuration must permit modification.

Postconditions

1. The memory area of the victim's process containing executable content is modified, i.e., it then contains executable content of the attacker's choice.

Architectural structures
Processes, memory, access control configuration, files.

Metrics
M_2 Limitation of number of executable components (table 3.4 on page 61) focuses on reducing the number of entry points for an attacker, M_4 Percentage of protected intermediate storage components (table 3.6 on page 62) also focuses on reducing the number of entry points for an attacker, M_5 Percentage of access control instrumentation (table 3.7 on page 63) focuses on protection of all entry points to a component, M_6 Conformity of access permissions (table 3.8 on page 63) focuses on identical permissions to ease correct administration, M_{10} Number of components with shared responsibility (server) (table 3.12 on page 65) focuses on reducing the number of targets acting as a force multiplier.

Specification in Z Observation is performed by an adversarial process. This process must be among the running processes.

$$
\begin{array}{l}
\textbf{schema } AttackModifyCodeInMemory_{Prologue} \\
\quad Processes \\
\quad AttackingProcess : HANDLE \\
\hline
\quad AttackingProcess \in runningProcesses
\end{array}
$$

The victim process is one of the running processes of the victim, i.e., the local human user's.

$$
\begin{array}{|l}
\textit{FindVictimProcess} \\
\hline
\textit{Processes} \\
\textit{VictimProcess} : \textit{HANDLE} \\
\hline
\textit{VictimProcess} \in \textit{runningProcesses} \\
\textit{processSubject}\ \textit{VictimProcess} = \textit{SUBJECT_VICTIM} \\
\hline
\end{array}
$$

The targeted memory area must belong to the victim process.

$$
\begin{array}{|l}
\textit{FindVictimMemoryArea} \\
\hline
\textit{Processes} \\
\textit{VictimProcess} : \textit{HANDLE} \\
\textit{VictimMemoryArea} : \textit{MemoryArea} \\
\hline
(\textit{VictimProcess}, \textit{VictimMemoryArea}) \in \textit{processMemory} \\
\hline
\end{array}
$$

The memory area must also contain executable code.

$$
\begin{array}{|l}
\textit{IsVictimMemoryAreaExecutable} \\
\hline
\textit{VictimMemoryArea} : \textit{MemoryArea} \\
\hline
\textit{VictimMemoryArea.content} \in \textit{CONTENT_EXECUTABLE} \\
\hline
\end{array}
$$

$$
\begin{array}{l}
\textit{AttackModifyCodeInMemory} == \\
\quad [\textit{AttackModifyCodeInMemory}_{\textit{Prologue}} \wedge \\
\quad \textit{FindVictimProcess} \wedge \textit{FindVictimMemoryArea} \wedge \\
\quad \textit{IsVictimMemoryAreaExecutable} \wedge \\
\quad \textit{UpdateMemory} \mid \textit{process}? = \textit{AttackingProcess} \wedge \\
\quad \textit{memoryHandle}? = \textit{VictimMemoryArea.handle} \wedge \\
\quad \textit{newMemoryData}? \in \textit{CONTENT_ATTACKERS_CHOICE}]
\end{array}
$$

Necessary attacker capabilities
Attack initiation capability: Attacker needs to be able to launch attack at own discretion
Available time to attack: Attacker needs to be able to carry out attack at least once per user session
Attack variation: Attacker needs not to be able to customise attack
Influence on user: Attacker needs not to be able to influence the user

6.1.2.2 Modify stored code module

Attack description The attacker tries to modify, delete, or replace a stored code module. The attacker has to be able to identify a code module of the victim. Success of the attack depends on the code module existing and its modification by the attacking process not being restricted by access rights. Only the interfaces of the operating system intended for modification of code modules are used.

Preconditions

1. The attacker must be able to find stored code modules a victim.
2. The code module must exist.
3. The access control configuration must permit modification.

Postconditions

1. The code module is modified, i.e., it contains executable content of the attacker's choice.

Architectural structures
Attacking process, data component code module, access control configuration.

Metrics
M_1 Centralisation of executable distribution sources (table 3.3 on page 60) focuses on reducing the number of entry points for an attacker, M_2 Limitation of number of executable components (table 3.4 on page 61) also focuses on reducing the number of entry points for an attacker, M_3 Percentage of protected executables (table 3.5 on page 62) focuses on reducing the vulnerability of possible entry points for an attacker, M_4 Percentage of protected intermediate storage components (table 3.6 on page 62) also focuses on reducing the number of entry points for an attacker, M_5 Percentage of access control instrumentation (table 3.7 on page 63) focuses on protection of all entry points to a component, M_6 Conformity of access permissions (table 3.8 on page 63) focuses on identical permissions to ease correct administration, M_7 Percentage of logged invocations (table 3.9 on page 64) focuses on detection/monitoring of attacks.

Specification in Z Modifying a stored code module is similar to modifying a stored data component directly, one difference being the content of the modified component being executable.

IsStoredDataComponentExecutable

$$
\begin{array}{l}
FileSystem \\
DataComponent : COMPNAME \\
\hline
\exists f : File \bullet \\
\quad fileToComponent\ f = DataComponent \land \\
\quad f.content \in CONTENT_EXECUTABLE
\end{array}
$$

$$
\begin{aligned}
&\mathit{AttackModifyStoredCodeModule} == \\
&\quad [\mathit{AttackModifyStoredDataComponent} \land \\
&\quad \mathit{IsStoredDataComponentExecutable} \mid \\
&\quad \mathit{DataComponent} \in (\mathrm{ran}\ \mathit{fileToComponent}) \land \\
&\quad \mathit{filecontent}? \in \mathit{CONTENT_EXECUTABLE}]
\end{aligned}
$$

Necessary attacker capabilities
Attack initiation capability: Attacker needs to be able to launch attack at own discretion
Available time to attack: Attacker needs to be able to carry out attack at least once per user session
Attack variation: Attacker needs not to be able to customise attack
Influence on user: Attacker needs not to be able to influence the user

6.1.2.3 Add stored code module

Attack description The attacker tries to add a new executable module with code of the attacker's choice to a container. The victim process executes components in the container and, hence, executes the added executable component. Success of the attack depends on adding the code module not being restricted by access rights. Only the interfaces of the operating system intended for adding a code module to a container are used.

Preconditions

1. The attacker must be able to find a container that is referenced by the victim.
2. The victim must execute code modules stored in the container.
3. The access control configuration must permit adding components to the container.

Postconditions

1. The container holds a new code module of the attacker's choice that can be executed by the victim process.

Architectural structures
Container, data storage, reference, processes.

Metrics
M_1 Centralisation of executable distribution sources (table 3.3 on page 60) focuses on reducing the number of entry points for an attacker, M_2 Limitation of number of executable components (table 3.4 on page 61) also focuses on reducing the number of entry points for an attacker, M_5 Percentage of access control instrumentation (table 3.7 on page 63) focuses on protection of all entry points to a component, M_6 Conformity of access permissions (table 3.8 on page 63) focuses on identical permissions to ease correct administration, M_7 Percentage of logged invocations (table 3.9 on page 64) focuses on detection/monitoring of attacks, M_{11} Number of components with multiple executable extensions (table 3.13 on page 66) focuses on reducing the vulnerability of possible entry points for an attacker.

Specification in Z An adversarial process must be among the running processes to perform the attack.

$$
\begin{array}{l}
\textit{AttackAddStoredCodeModule}_{\textit{Prologue}} \\
\hline
\textit{Processes} \\
\textit{AttackingProcess} : \textit{HANDLE} \\
\hline
\textit{AttackingProcess} \in \textit{runningProcesses}
\end{array}
$$

First, containers are identified in the file system.

$$
\begin{array}{l}
\textbf{FindContainer} \\
\hline
Configuration \\
FileSystem \\
ContainerComponent : COMPNAME \\
\hline
(components\ ContainerComponent).componentType = CTDATA \\
\exists\, connname : CONNNAME;\ cp : PORT;\ fd : Folder \bullet \\
\quad (connectors\ connname).connectionType = CNCONTAINEDBY \land \\
\quad ((connname, ROCONTAINER), (ContainerComponent, cp)) \in attachment \land \\
\quad folderToComponent\ fd = ContainerComponent
\end{array}
$$

The selection of containers is restricted to those being referenced as containing executable modules.

$$
\begin{array}{l}
\textbf{IsContainerUsedInExecutionReference} \\
\hline
Configuration \\
Processes \\
FileSystem \\
ContainerComponent : COMPNAME \\
\hline
\exists\, proc : HANDLE;\ proccomp : COMPNAME \bullet \\
\exists\, ReferenceComponent : COMPNAME \bullet \\
\exists\, dtconn : CONNNAME;\ refconn : CONNNAME \bullet \\
\exists\, prsp : PORT;\ prtp : PORT;\ rcsp : PORT;\ rctp : PORT \bullet \\
\quad (fileToComponent\ (processFile\ proc)) = proccomp \land \\
\quad ((connectors\ dtconn).connectionType = CNLINKEDEXEC \lor \\
\quad (connectors\ dtconn).connectionType = CNEXECUTE) \land \\
\quad ((dtconn, ROEXECUTOR), (proccomp, prsp)) \in attachment \land \\
\quad ((dtconn, ROEXECUTED), (ReferenceComponent, prtp)) \in attachment \land \\
\quad ((refconn, ROREFERENCESOURCE), \\
\qquad (ReferenceComponent, rcsp)) \in attachment \land \\
\quad ((refconn, ROREFERENCETARGET), \\
\qquad (ContainerComponent, rctp)) \in attachment \land \\
\quad \exists\, ownercomp : COMPNAME;\ oconn : CONNNAME;\ onp : PORT;\ odp : PORT \bullet \\
\qquad (components\ ownercomp).componentType = CTSUBJECTVICTIM \land \\
\qquad (connectors\ oconn).connectionType = CNOWNER \land \\
\qquad ((oconn, ROOWNER), (ownercomp, onp)) \in attachment \land \\
\qquad ((oconn, ROOWNED), (proccomp, odp)) \in attachment
\end{array}
$$

A new executable module is added to the container.

$$
\begin{array}{l}
\textit{AddExecutableFileInFolder} \\
\hline
\Delta \textit{AccessControlPolicy} \\
\textit{CreateFile} \\
\textit{ContainerComponent} : \textit{COMPNAME} \\
\hline
\exists \textit{fn} : \textit{PATHNAME} \bullet \\
\quad \textit{fn} = \textit{newFilename}? \land \\
\quad \textit{folderToComponent}\ (\textit{nodeFolder}\ (\textit{nodeByFullName}\ (\textit{front fn}))) = \\
\quad\quad \textit{ContainerComponent} \land \\
\quad \textit{fn} \notin \operatorname{dom}\ \textit{nodeByFullName} \land \\
\quad (\textit{SUBJECT_VICTIM}, \{\textit{ACCESS_INVOKE}\}) \in \textit{newFileAccessMode}?
\end{array}
$$

The attack consists of finding an appropriate container and adding an executable module to it.

$$
\begin{array}{l}
\textit{AttackAddStoredCodeModule} == \\
\quad [\textit{AttackAddStoredCodeModule}_{\textit{Prologue}} \land \\
\quad \textit{FindContainer} \land \\
\quad \textit{IsContainerUsedInExecutionReference} \land \\
\quad \textit{AddExecutableFileInFolder} \mid \textit{requestingSubject}? = \textit{processSubject AttackingProcess}]
\end{array}
$$

Necessary attacker capabilities
Attack initiation capability: Attacker needs to wait for the victim process to execute the added code module
Available time to attack: Attacker needs to be able to carry out attack at least once per user session
Attack variation: Attacker needs not to be able to customise attack
Influence on user: Attacker needs not to be able to influence the user

6.1.2.4 Modify reference to stored code module

Attack description Modify reference to existing data component, labelled as executable, to point to a different (existing) executable data component, belonging to the attacked process or another process not under attacker's control (rearranging the execution path) or to the attacker's process (controlling what code is to be executed in detail).

Preconditions

1. The attacker must be able to find a reference used by the victim.
2. The victim must execute code via the reference.
3. The access control configuration must permit modifying the reference.

Postconditions

1. The reference points to a different executable module that contains executable data of the attacker's choice.

Architectural structures
Reference, data storage, processes.

Metrics
M_2 Limitation of number of executable components (table 3.4 on page 61) focuses on reducing the number of entry points for an attacker, M_3 Percentage of protected executables (table 3.5 on page 62) focuses on reducing the vulnerability of possible entry points for an attacker, M_5 Percentage of access control instrumentation (table 3.7 on page 63) focuses on protection of all entry points to a component, M_6 Conformity of access permissions (table 3.8 on page 63) focuses on identical permissions to ease correct administration, M_7 Percentage of logged invocations (table 3.9 on page 64) focuses on detection/monitoring of attacks, M_{11} Number of components with multiple executable extensions (table 3.13 on page 66) focuses on reducing the vulnerability of possible entry points for an attacker.

Specification in Z An adversarial process must be among the running processes to perform the attack.

```
┌─ AttackModifyReferenceToStoredCodeModule_Prologue ──────
│ Processes
│ AttackingProcess : HANDLE
├──────────
│ AttackingProcess ∈ runningProcesses
└──────────────────────────────────────────
```

A reference needs to be identified.

$$
\begin{array}{l}
\textit{FindReference} \\
\hline
\textit{Configuration} \\
\textit{FileSystem} \\
\textit{ReferenceComponent} : \textit{COMPNAME} \\
\textit{TargetComponent} : \textit{COMPNAME} \\
\hline
(\textit{components ReferenceComponent}).\textit{componentType} = \textit{CTDATA} \\
(\textit{components TargetComponent}).\textit{componentType} = \textit{CTDATA} \\
\exists\, \textit{connname} : \textit{CONNNAME};\ \textit{cp} : \textit{PORT};\ f : \textit{File} \bullet \\
\quad ((\textit{connectors connname}).\textit{connectionType} = \textit{CNREFERENCERULESTATIC} \lor \\
\quad (\textit{connectors connname}).\textit{connectionType} = \textit{CNREFERENCERULESEARCHORDER}) \land \\
\quad ((\textit{connname}, \textit{ROREFERENCESOURCE}), (\textit{ReferenceComponent}, \textit{cp})) \in \textit{attachment} \land \\
\quad ((\textit{connname}, \textit{ROREFERENCETARGET}), (\textit{fileToComponent}\ f, \textit{cp})) \in \textit{attachment}
\end{array}
$$

The set of references is restricted to those of the victim process pointing to executable modules.

$$
\begin{array}{l}
\textit{IsReferenceToCodeModule} \\
\hline
\textit{Configuration} \\
\textit{Processes} \\
\textit{FileSystem} \\
\textit{ReferenceComponent} : \textit{COMPNAME} \\
\hline
\exists\, \textit{proc} : \textit{HANDLE};\ \textit{proccomp} : \textit{COMPNAME} \bullet \\
\exists\, \textit{CodeComponent} : \textit{COMPNAME} \bullet \\
\exists\, \textit{dtconn} : \textit{CONNNAME};\ \textit{refconn} : \textit{CONNNAME} \bullet \\
\exists\, \textit{prsp} : \textit{PORT};\ \textit{prtp} : \textit{PORT};\ \textit{rcsp} : \textit{PORT};\ \textit{rctp} : \textit{PORT} \bullet \\
\quad (\textit{fileToComponent}\ (\textit{processFile proc})) = \textit{proccomp} \land \\
\quad ((\textit{connectors dtconn}).\textit{connectionType} = \textit{CNLINKEDEXEC} \lor \\
\quad (\textit{connectors dtconn}).\textit{connectionType} = \textit{CNEXECUTE}) \land \\
\quad ((\textit{dtconn}, \textit{ROEXECUTOR}), (\textit{proccomp}, \textit{prsp})) \in \textit{attachment} \land \\
\quad ((\textit{dtconn}, \textit{ROEXECUTED}), (\textit{ReferenceComponent}, \textit{prtp})) \in \textit{attachment} \land \\
\quad ((\textit{refconn}, \textit{ROREFERENCESOURCE}), (\textit{ReferenceComponent}, \textit{rcsp})) \in \textit{attachment} \land \\
\quad ((\textit{refconn}, \textit{ROREFERENCETARGET}), (\textit{CodeComponent}, \textit{rctp})) \in \textit{attachment} \land \\
\quad \exists\, \textit{ownercomp} : \textit{COMPNAME};\ \textit{oconn} : \textit{CONNNAME};\ \textit{onp} : \textit{PORT};\ \textit{odp} : \textit{PORT} \bullet \\
\qquad (\textit{components ownercomp}).\textit{componentType} = \textit{CTSUBJECTVICTIM} \land \\
\qquad (\textit{connectors oconn}).\textit{connectionType} = \textit{CNOWNER} \land \\
\qquad ((\textit{oconn}, \textit{ROOWNER}), (\textit{ownercomp}, \textit{onp})) \in \textit{attachment} \land \\
\qquad ((\textit{oconn}, \textit{ROOWNED}), (\textit{proccomp}, \textit{odp})) \in \textit{attachment}
\end{array}
$$

The reference is modified to point to a component of the attacker's choice.

$$
\begin{array}{l}
\textit{ModifyReference} \\
\hline
\textit{Configuration} \\
\textit{UpdateFile} \\
\textit{ReferenceComponent} : \textit{COMPNAME} \\
\hline
\textit{filecontent?} \in \textit{CONTENT_ATTACKERS_CHOICE} \\
\exists f : \textit{File} \bullet \\
\quad \textit{fileByFullName filename?} = f \land \\
\quad \textit{fileToComponent } f = \textit{ReferenceComponent}
\end{array}
$$

The attack consists of finding a reference to an executable module and the modifying the reference.

$$
\begin{array}{l}
\textit{AttackModifyReferenceToStoredCodeModule} == \\
\quad [\textit{AttackModifyReferenceToStoredCodeModule}_{\textit{Prologue}} \land \\
\quad \textit{FindReference} \land \\
\quad \textit{IsReferenceToCodeModule} \land \\
\quad \textit{ModifyReference} \mid \textit{requestingSubject?} = \textit{processSubject AttackingProcess}]
\end{array}
$$

Necessary attacker capabilities
Attack initiation capability: Attacker needs to wait for the victim process to execute the referenced code module
Available time to attack: Attacker needs to be able to carry out attack at least once per user session
Attack variation: Attacker needs not to be able to customise attack
Influence on user: Attacker needs not to be able to influence the user

6.1.3 Violation of integrity of parameters

The attacker tries to modify or replace a component containing parameters imported by connectors to code components. The attacker has to be able to identify a component containing parameters for the victim process. Success of the attack depends on the parameter component existing and its modification by the attacking process not being restricted by access rights.

6.1.3.1 Initiate communication with component and send data

Attack description Start communication via IPC mechanism and send data from attacker's process to attacked process. May need to create/start attacked process, based on stored data components labelled as executable and as belonging to attacked process.

Preconditions

1. The attacker must be able to find an IPC interface of the victim process.

2. Data transferred via the interface must be interpreted as a parameter by the victim.

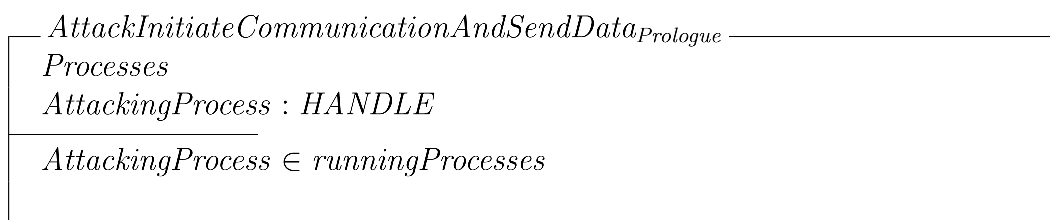
Postconditions

1. The victim process bases its execution on parameters of the attacker's choice.

Architectural structures
Data storage, data transfer, parameters, processes.

Metrics
M_7 Percentage of logged invocations (table 3.9 on page 64) focuses on detection/monitoring of attacks, M_8 Percentage of authenticity/integrity preserving connectors (table 3.10 on page 64) focuses on prevention of attacks.

Specification in Z An adversarial process must be among the running processes to perform the attack.



The attacker needs to determine if the victim process possesses IPC interfaces.

$$
\begin{array}{l}
\textbf{FindExecutableFileOfVictimProcessModules} \\
\hline
Configuration \\
FileSystem \\
Processes \\
LinkedLibraries \\
victimProcess : HANDLE \\
\hline
\exists\, executableFile : File \bullet \\
\quad (\exists\, ownercomp : COMPNAME;\ oconn : CONNNAME;\ onp : PORT;\ odp : PORT \bullet \\
\quad\quad (components\ ownercomp).componentType = CTSUBJECTVICTIM \land \\
\quad\quad (connectors\ oconn).connectionType = CNOWNER \land \\
\quad\quad ((oconn, ROOWNER), (ownercomp, onp)) \in attachment \land \\
\quad\quad ((oconn, ROOWNED), (fileToComponent\ executableFile, odp)) \in attachment) \land \\
\quad ((processFile\ victimProcess = executableFile) \lor \\
\quad \exists\, l : HANDLE \bullet \\
\quad\quad (victimProcess, l) \in processLibraries \land \\
\quad\quad libraryFile\ l = executableFile)
\end{array}
$$

Data received via an IPC mechanism must be interpreted as a parameter to become a target for the adversary.

$$
\begin{array}{l}
\textbf{FindIPCComponentOfVictimProcess} \\
\hline
Configuration \\
FileSystem \\
Processes \\
FindExecutableFileOfVictimProcessModules \\
\hline
\exists\, proccomp : COMPNAME;\ ipcdatacomp : COMPNAME \bullet \\
\exists\, ipcconn : CONNNAME;\ paramconn : CONNNAME \bullet \\
\exists\, procipcp : PORT;\ ipcdatap : PORT \bullet \\
\quad fileToComponent\ (processFile\ victimProcess) = proccomp \land \\
\quad (components\ ipcdatacomp).componentType = CTDATA \land \\
\quad (connectors\ ipcconn).connectionType = CNDATATRANSFER \land \\
\quad (connectors\ paramconn).connectionType = CNPARAM \land \\
\quad ((ipcconn, RODATATARGET), (proccomp, procipcp)) \in attachment \land \\
\quad ((ipcconn, RODATASOURCE), (ipcdatacomp, ipcdatap)) \in attachment \land \\
\quad ((paramconn, ROPARAMSOURCE), (proccomp, procipcp)) \in attachment
\end{array}
$$

The attack consists of finding a relevant IPC interface and then sending data of the attacker's choice via the interface.

$$\begin{aligned}
&\mathit{AttackInitiateCommunicationAndSendData} == \\
&\quad [\mathit{AttackInitiateCommunicationAndSendData}_{\mathit{Prologue}} \land \\
&\quad \mathit{FindIPCComponentOfVictimProcess} \land \\
&\quad \mathit{InvokeIPC} \mid \mathit{sourceProcess} = \mathit{AttackingProcess} \land \\
&\quad \mathit{targetProcess} = \mathit{victimProcess} \land \\
&\quad \mathit{sendData} \in \mathit{CONTENT_ATTACKERS_CHOICE}]
\end{aligned}$$

Necessary attacker capabilities
Attack initiation capability: Attacker needs to be able to launch attack at own discretion
Available time to attack: Attacker needs to be able to carry out attack several times per user session
Attack variation: Attacker needs not to be able to customise attack
Influence on user: Attacker needs not to be able to influence the user

6.1.3.2 Respond to component's communication request and send data

Attack description Wait for attacked process to use IPC mechanisms of attacker's process. Attacker's process may need to be created first. Then send data to attacked process via IPC mechanism. Alternatively, have data avilable for the attacked process to request.

Preconditions

1. The attacker must be able to position itself as the target of a communication request by the victim process via an IPC interface.
2. Data transferred via the interface must be interpreted as a parameter by the victim.

Postconditions

1. The victim process bases its execution on parameters of the attacker's choice.

Architectural structures
Data storage, data transfer, parameters, processes.

Metrics
M_7 Percentage of logged invocations (table 3.9 on page 64) focuses on detection/monitoring of attacks, M_8 Percentage of authenticity/integrity preserving connectors (table 3.10 on page 64) focuses on prevention of attacks.

Specification in Z An adversarial process must be among the running processes to perform the attack and the process must be able to become a target of IPC communication with the victim process.

$$
\begin{array}{l}
\textit{FindExecutableFileOfAdversarialProcessModules} \\
\hline
\textit{Configuration} \\
\textit{FileSystem} \\
\textit{Processes} \\
\textit{LinkedLibraries} \\
\textit{adversarialProcess} : \textit{HANDLE} \\
\hline
\exists\, \textit{executableFile} : \textit{File} \bullet \\
\quad \textit{adversarialProcess} \in \textit{runningProcesses} \land \\
\quad (\exists\, \textit{ownercomp} : \textit{COMPNAME};\ \textit{oconn} : \textit{CONNNAME};\ \textit{onp} : \textit{PORT};\ \textit{odp} : \textit{PORT} \bullet \\
\qquad ((\textit{components ownercomp}).\textit{componentType} = \textit{CTSUBJECTADVERSARY} \lor \\
\qquad (\textit{components ownercomp}).\textit{componentType} = \textit{CTSUBJECTUNSPECIFIED}) \land \\
\qquad (\textit{connectors oconn}).\textit{connectionType} = \textit{CNOWNER} \land \\
\qquad ((\textit{oconn}, \textit{ROOWNER}), (\textit{ownercomp}, \textit{onp})) \in \textit{attachment} \land \\
\qquad ((\textit{oconn}, \textit{ROOWNED}), (\textit{fileToComponent executableFile}, \textit{odp})) \in \textit{attachment}) \land \\
\quad ((\textit{processFile adversarialProcess} = \textit{executableFile}) \lor \\
\quad \exists\, l : \textit{HANDLE} \bullet \\
\qquad (\textit{adversarialProcess}, l) \in \textit{processLibraries} \land \\
\qquad \textit{libraryFile}\ l = \textit{executableFile})
\end{array}
$$

Data communicated to the victim process must be interpreted as parameters.

$$
\begin{array}{l}
\textit{FindIPCComponentOfAdversarialProcess} \\
\hline
\textit{Configuration} \\
\textit{FileSystem} \\
\textit{Processes} \\
\textit{FindExecutableFileOfAdversarialProcessModules} \\
\hline
\exists\, proccomp : COMPNAME;\ ipcdatacomp : COMPNAME \bullet \\
\exists\, ipcconn : CONNNAME;\ paramconn : CONNNAME \bullet \\
\exists\, procipcp : PORT;\ ipcdatap : PORT \bullet \\
\quad fileToComponent\ (processFile\ adversarialProcess) = proccomp \land \\
\quad (components\ ipcdatacomp).componentType = CTDATA \land \\
\quad (connectors\ ipcconn).connectionType = CNDATATRANSFER \land \\
\quad (connectors\ paramconn).connectionType = CNPARAM \land \\
\quad ((ipcconn, RODATATARGET), (proccomp, procipcp)) \in attachment \land \\
\quad ((ipcconn, RODATASOURCE), (ipcdatacomp, ipcdatap)) \in attachment \land \\
\quad ((paramconn, ROPARAMSOURCE), (proccomp, procipcp)) \in attachment
\end{array}
$$

The attack consists of positioning the adversarial process as target of IPC communication initiated by the victim process and providing data of the attacker's choice on request.

$$
\begin{array}{l}
AttackRespondCommunicationAndSendData == \\
\quad [FindExecutableFileOfVictimProcessModules \land \\
\quad FindIPCComponentOfAdversarialProcess \land \\
\quad InvokeIPC \mid sourceProcess = victimProcess \land \\
\quad targetProcess = adversarialProcess \land \\
\quad sendData \in \mathrm{dom}(processIPCResponse\ victimProcess) \land \\
\quad \mathrm{ran}(processIPCResponse\ adversarialProcess) \subseteq CONTENT_ATTACKERS_CHOICE]
\end{array}
$$

Necessary attacker capabilities

Attack initiation capability: Attacker needs to wait for the victim process to execute the referenced code module

Available time to attack: Attacker needs to be able to carry out attack several times per user session

Attack variation: Attacker needs to be able to customise the attack, but does not need to simulate human behaviour

Influence on user: Attacker needs not to be able to influence the user

6.1.3.3 Modify stored data component containing parameters

Attack description Like 6.1.1.1, but data component is labelled as containing parameters/labelled as content being used as parameter(s).

Preconditions

1. The attacker must be able to find a victim's stored parameters (any).
2. The component containing the parameters must exist.
3. The access control configuration must permit modification.

Postconditions

1. The parameters are modified, i.e., they are of the attacker's choice.

Architectural structures
Attacking process, stored parameters, access control configuration.

Metrics
M_5 Percentage of access control instrumentation (table 3.7 on page 63) focuses on protection of all entry points to a component, M_6 Conformity of access permissions (table 3.8 on page 63) focuses on identical permissions to ease correct administration.

Specification in Z Modifying a stored code module is similar to modifying a stored data component directly, one difference being the content of the modified component being used as parameters by the victim.

$$
\begin{array}{l}
\textit{StoredDataComponentContainsParameters} \\
\hline
\textit{Configuration} \\
\textit{FileSystem} \\
\textit{DataComponent} : \textit{COMPNAME} \\
\hline
(\textit{components}\ \textit{DataComponent}).\textit{componentType} = \textit{CTDATA} \\
(\exists\, \textit{connname} : \textit{CONNNAME} \bullet \\
\exists\, \textit{ocn} : \textit{COMPNAME} \bullet \exists\, \textit{ocp} : \textit{PORT} \bullet \exists\, \textit{cp} : \textit{PORT} \bullet \\
\quad (\textit{connectors}\ \textit{connname}).\textit{connectionType} = \textit{CNOWNER} \land \\
\quad (\textit{components}\ \textit{ocn}).\textit{componentType} = \textit{CTSUBJECTVICTIM} \land \\
\quad ((\textit{connname}, \textit{ROSUBJECT}), (\textit{ocn}, \textit{ocp})) \in \textit{attachment} \land \\
\quad ((\textit{connname}, \textit{ROOBJECT}), (\textit{DataComponent}, \textit{cp})) \in \textit{attachment}) \\
(\exists\, \textit{connname} : \textit{CONNNAME} \bullet \\
\exists\, \textit{ocn} : \textit{COMPNAME} \bullet \exists\, \textit{ocp} : \textit{PORT} \bullet \exists\, \textit{cp} : \textit{PORT} \bullet \\
\quad (\textit{connectors}\ \textit{connname}).\textit{connectionType} = \textit{CNPARAM} \land \\
\quad ((\textit{connname}, \textit{ROPARAMPROCESSOR}), (\textit{ocn}, \textit{ocp})) \in \textit{attachment} \land \\
\quad ((\textit{connname}, \textit{ROPARAMSOURCE}), (\textit{DataComponent}, \textit{cp})) \in \textit{attachment}) \\
\exists\, f : \textit{File} \bullet \textit{fileToComponent}\ f = \textit{DataComponent}
\end{array}
$$

The attack consists of identifying a data component being used as parameters and then modifying the component with data of the attacker's choice.

$$\begin{array}{l} AttackModifyStoredParameters == \\ \quad [AttackModifyStoredDataComponent \land \\ \quad StoredDataComponentContainsParameters \mid \\ \quad DataComponent \in (\mathrm{ran}\ fileToComponent)] \end{array}$$

Necessary attacker capabilities
Attack initiation capability: Attacker needs to be able to launch attack at own discretion
Available time to attack: Attacker needs to be able to carry out attack at least once per user session
Attack variation: Attacker needs not to be able to customise attack
Influence on user: Attacker needs not to be able to influence the user

6.1.3.4 Modify reference to stored data component containing parameters

Attack description The attacker tries to modify a reference to an existing data component labelled as being used as parameter(s).

Preconditions

1. The attacker must be able to find a reference used by the victim.

2. The victim must interpret data received via the reference as parameters.

3. The access control configuration must permit modifying the reference.

Postconditions

1. The reference points to a different data component containing data of the attacker's choice.

Architectural structures
Reference, data storage, parameters, processes.

Metrics
M_4 Percentage of protected intermediate storage components (table 3.6 on page 62) focuses on reducing the number of entry points for an attacker, M_5 Percentage of access control instrumentation (table 3.7 on page 63) focuses on protection of all entry points to a component, M_6 Conformity of access permissions (table 3.8 on page 63) focuses on identical permissions to ease correct administration, M_8 Percentage of authenticity/integrity preserving connectors (table 3.10 on page 64) focuses on prevention of attacks.

Specification in Z An adversarial process must be among the running processes to perform the attack.

```
┌─ AttackModifyReferenceToStoredParameters_Prologue ──────
│ Processes
│ AttackingProcess : HANDLE
├──────────
│ AttackingProcess ∈ runningProcesses
│
└─────────────────────────────────────
```

The attacker needs to identify a reference component that points to a data component that is being treated as parameters by the victim process.

$$
\begin{array}{l}
\textit{IsReferenceToParameters} \\
\hline
\textit{Configuration} \\
\textit{Processes} \\
\textit{FileSystem} \\
\textit{ReferenceComponent} : \textit{COMPNAME} \\
\hline
\exists\, \textit{proc} : \textit{HANDLE};\ \textit{proccomp} : \textit{COMPNAME} \bullet \\
\exists\, \textit{ParamComponent} : \textit{COMPNAME} \bullet \\
\exists\, \textit{dtconn} : \textit{CONNNAME};\ \textit{refconn} : \textit{CONNNAME};\ \textit{paramconn} : \textit{CONNNAME} \bullet \\
\exists\, \textit{prsp} : \textit{PORT};\ \textit{prtp} : \textit{PORT};\ \textit{rcsp} : \textit{PORT};\ \textit{rctp} : \textit{PORT} \bullet \\
\quad (\textit{fileToComponent}\ (\textit{processFile}\ \textit{proc})) = \textit{proccomp} \land \\
\quad (\textit{connectors}\ \textit{paramconn}).\textit{connectionType} = \textit{CNPARAM} \land \\
\quad ((\textit{paramconn}, \textit{ROPARAMSOURCE}), (\textit{proccomp}, \textit{prsp})) \in \textit{attachment} \land \\
\quad (\textit{connectors}\ \textit{dtconn}).\textit{connectionType} = \textit{CNDATATRANSFER} \land \\
\quad ((\textit{dtconn}, \textit{RODATATARGET}), (\textit{proccomp}, \textit{prsp})) \in \textit{attachment} \land \\
\quad ((\textit{dtconn}, \textit{RODATASOURCE}), (\textit{ReferenceComponent}, \textit{prtp})) \in \textit{attachment} \land \\
\quad ((\textit{refconn}, \textit{ROREFERENCESOURCE}), (\textit{ReferenceComponent}, \textit{rcsp})) \in \textit{attachment} \land \\
\quad ((\textit{refconn}, \textit{ROREFERENCETARGET}), (\textit{ParamComponent}, \textit{rctp})) \in \textit{attachment} \land \\
\quad \exists\, \textit{ownercomp} : \textit{COMPNAME};\ \textit{oconn} : \textit{CONNNAME};\ \textit{onp} : \textit{PORT};\ \textit{odp} : \textit{PORT} \bullet \\
\quad\quad (\textit{components}\ \textit{ownercomp}).\textit{componentType} = \textit{CTSUBJECTVICTIM} \land \\
\quad\quad (\textit{connectors}\ \textit{oconn}).\textit{connectionType} = \textit{CNOWNER} \land \\
\quad\quad ((\textit{oconn}, \textit{ROOWNER}), (\textit{ownercomp}, \textit{onp})) \in \textit{attachment} \land \\
\quad\quad ((\textit{oconn}, \textit{ROOWNED}), (\textit{proccomp}, \textit{odp})) \in \textit{attachment}
\end{array}
$$

The attack consists of identifying an appropriate reference and then modifying the reference with content of the attacker's choice.

$$
\begin{array}{l}
\textit{AttackModifyReferenceToStoredParameters} == \\
\quad [\textit{AttackModifyReferenceToStoredParameters}_{\textit{Prologue}} \land \\
\quad \textit{FindReference} \land \\
\quad \textit{IsReferenceToParameters} \land \\
\quad \textit{ModifyReference} \mid \textit{requestingSubject?} = \textit{processSubject}\ \textit{AttackingProcess}]
\end{array}
$$

Necessary attacker capabilities

Attack initiation capability: Attacker needs to wait for the victim process to obtain data from the referenced data component

Available time to attack: Attacker needs to be able to carry out attack at least once per user session

Attack variation: Attacker needs not to be able to customise attack

Influence on user: Attacker needs not to be able to influence the user

6.1.3.5 Simulate user input

Attack description Modify/create user interface input data, i.e., data components labelled as *user input*, depending on access permissions to user interface objects and their containers (if any).

Preconditions

1. Attacker must be able to identify input queues used by victim process.
2. The access control configuration must permit modifying the queue.

Postconditions

1. The input queue of the victim process contains data of the attacker's choice that is interpreted as having originated from the local human user.

Architectural structures
Process, user input.

Metrics
M_{12} Percentage of trusted path connectors (table 3.14 on page 67) focuses on prevention and detection of attacks on a user interface.

Specification in Z An adversarial process must be among the running processes to perform the attack.

$$
\begin{array}{l}
\textit{AttackSimulateUserInput}_{Prologue} \\
\hline
\textit{Processes} \\
\textit{AttackingProcess} : \textit{HANDLE} \\
\hline
\textit{AttackingProcess} \in \textit{runningProcesses}
\end{array}
$$

An input queue belonging to the victim process must be identified.

$$
\begin{array}{l}
\textit{FindVictimUIInputQueue} \\
\hline
\textit{Processes} \\
\textit{victimUIInputQueue} : \textit{HANDLE} \\
\hline
\exists\, \textit{proc} : \textit{HANDLE} \bullet \\
\quad \textit{processSubject}\ \textit{proc} = \textit{SUBJECT_VICTIM} \land \\
\quad (\textit{proc}, \textit{victimUIInputQueue}) \in \textit{processInput}
\end{array}
$$

The attack consists of finding an input queue of the victim process and then appending data of the attacker's choice to the queue.

$$
\begin{aligned}
&\mathit{AttackSimulateUserInput} == \\
&\quad [\mathit{AttackSimulateUserInput}_{\mathit{Prologue}} \land \\
&\quad \mathit{FindVictimUIInputQueue} \land \\
&\quad \mathit{WriteUIInput} \mid \mathit{inputDevice} = \mathit{victimUIInputQueue} \land \\
&\quad \mathit{requestingSubject}? = \mathit{processSubject}\ \mathit{AttackingProcess} \land \\
&\quad \mathit{UIData} \in \mathit{CONTENT_ATTACKERS_CHOICE}]
\end{aligned}
$$

Necessary attacker capabilities
Attack initiation capability: Attacker needs to be able to launch attack at own discretion
Available time to attack: Attacker needs to be able to carry out attack several times per user session
Attack variation: Attacker needs to be able to customise the attack, and may need to be able to simulate human behaviour
Influence on user: Attacker needs not to be able to influence the user

6.1.4 Influence on user

The attacker tries to modify, delete, or replace information that the user bases a security-relevant decision on. This comprises user interface components as well as, e.g., logging data.

6.1.4.1 Modify user interface object

Attack description Modify/create user interface output data, i.e., data components labelled as *output to user*, depending on access permissions to user interface objects.

Preconditions

1. Attacker must be able to identify output devices used by victim process.
2. The access control configuration must permit modifying the device content.

Postconditions

1. The output device contains data of the attacker's choice that is used by the local human user in security-relevant decisions.

Architectural structures
Process, user output.

Metrics
M_{12} Percentage of trusted path connectors (table 3.14 on page 67) focuses on prevention and detection of attacks on a user interface.

Specification in Z An adversarial process must be among the running processes to perform the attack.

$$
\begin{array}{l}
\textit{AttackModifyUserInterfaceObject}_{\textit{Prologue}} \\
\hline
\textit{Processes} \\
\textit{AttackingProcess} : \textit{HANDLE} \\
\hline
\textit{AttackingProcess} \in \textit{runningProcesses}
\end{array}
$$

The attacker has to determine which output devices are used by the victim process to present security-relevant data to the local human user.

$$
\begin{array}{l}
\textit{FindVictimUIOutputDevice} \\
\hline
\textit{Configuration} \\
\textit{UIOutputDevices} \\
\textit{Processes} \\
\textit{victimUIOutputDevice} : \textit{HANDLE} \\
\hline
\exists\, \textit{proc} : \textit{HANDLE} \bullet \\
\quad \textit{processSubject proc} = \textit{SUBJECT_VICTIM} \wedge \\
\quad (\textit{proc}, \textit{victimUIOutputDevice}) \in \textit{processOutput} \wedge \\
\quad \exists\, \textit{uiocomp} : \textit{COMPNAME};\ \textit{uioconn} : \textit{CONNNAME};\ \textit{uiop} : \textit{PORT} \bullet \\
\quad\quad \textit{uiOutputDeviceToComponent victimUIOutputDevice} = \textit{uiocomp} \wedge \\
\quad\quad (((\textit{uioconn}, \textit{ROLOGDATATARGET}), (\textit{uiocomp}, \textit{uiop})) \in \textit{attachment} \vee \\
\quad\quad ((\textit{uioconn}, \textit{ROMONITORTARGET}), (\textit{uiocomp}, \textit{uiop})) \in \textit{attachment})
\end{array}
$$

The attack consists of finding the relevant output devices and then modifying their contents.

$$
\begin{array}{l}
\textit{AttackModifyUserInterfaceObject} == \\
\quad [\textit{AttackModifyUserInterfaceObject}_{\textit{Prologue}} \wedge \\
\quad \textit{FindVictimUIOutputDevice} \wedge \\
\quad \textit{WriteUIOutput} \mid \textit{outputDevice} = \textit{victimUIOutputDevice} \wedge \\
\quad \textit{requestingSubject?} = \textit{processSubject AttackingProcess} \wedge \\
\quad \textit{UIData} \in \textit{CONTENT_ATTACKERS_CHOICE}]
\end{array}
$$

Necessary attacker capabilities
Attack initiation capability: Attacker needs to wait for the local human user to determine the moment to start the attack
Available time to attack: Attacker needs to be able to carry out attack several times per user session, sometimes permanently
Attack variation: Attacker needs to be able to customise the attack, but does not need to be able to simulate human behaviour
Influence on user: Attacker needs to be able to influence the user

6.1.4.2 Modify stored data component examined by users as basis for decisions

Attack description Like 6.1.1.1, but data component labelled as *evidence/support for user decisions*, depending on access permissions to user interface objects and their containers (if any).

Preconditions

1. The attacker must be able to find a victim's stored data component (any).
2. The data component must exist.
3. The access control configuration must permit modification.

Postconditions

1. The data component is modified, i.e., it contains data of the attacker's choice.

Architectural structures
Attacking process, data component, access control configuration.

Metrics
M_5 Percentage of access control instrumentation (table 3.7 on page 63) focuses on protection of all entry points to a component, M_6 Conformity of access permissions (table 3.8 on page 63) focuses on identical permissions to ease correct administration.

Specification in *Z* Modifying a stored data component used in decision making is a special case of modifying an arbitrary stored data component directly. In the current version of our model we have logging data, monitoring data, and integrity verification data as data that security-relevant decisions can be based on.

$$
\begin{array}{l}
\textit{IsStoredDataComponentUsedInDecision} \\
\hline
\textit{Configuration} \\
\textit{DataComponent} : \textit{COMPNAME} \\
\hline
\exists\, \textit{dcconn} : \textit{CONNNAME};\ \textit{dcp} : \textit{PORT} \bullet \\
\quad ((\textit{dcconn}, \textit{ROLOGDATATARGET}), (\textit{DataComponent}, \textit{dcp})) \in \textit{attachment} \lor \\
\quad ((\textit{dcconn}, \textit{ROMONITORTARGET}), (\textit{DataComponent}, \textit{dcp})) \in \textit{attachment} \lor \\
\quad ((\textit{dcconn}, \textit{ROVERIFICATIONDATA}), (\textit{DataComponent}, \textit{dcp})) \in \textit{attachment}
\end{array}
$$

The attack consists of identifying a stored data component containing decision support data and then modifying the component.

$$
\begin{array}{l}
\textit{AttackModifyStoredDataComponentForDecisions} == \\
\quad \textit{AttackModifyStoredDataComponent} \land \\
\quad \textit{IsStoredDataComponentUsedInDecision}
\end{array}
$$

Necessary attacker capabilities
Attack initiation capability: Attacker needs to be able to launch attack at own discretion
Available time to attack: Attacker needs to be able to carry out attack at least once per user session
Attack variation: Attacker needs not to be able to customise attack
Influence on user: Attacker needs not to be able to influence the user

6.2 Security requirements

The metrics for *security requirements* form a lattice by construction. A value of the metric is a triple with its components taken from four inverse sequences, respectively. We recall the three axes from section 3.1 on page 46 and define the according sequences.

1. *Limit damage done to data integrity*: Attacks on the integrity of data items can remain undetected, can be logged, can be monitored and the user alerted, can be prevented.

$$
\begin{aligned}
&\mathit{GenericSecReq} ::= \\
&\quad \mathit{Undetected} \mid \mathit{Log} \mid \mathit{Alert} \mid \mathit{Prevent} \\
&\mathit{SecReqDataInt} == \mathit{GenericSecReq} \\
&\mathit{GenericSecReqLevel} == \{ \\
&\quad (\mathit{Undetected} \mapsto 1), (\mathit{Log} \mapsto 2), (\mathit{Alert} \mapsto 3), (\mathit{Prevent} \mapsto 4)\} \\
&\mathit{SecReqDataIntLevel} == \mathit{GenericSecReqLevel}
\end{aligned}
$$

2. *Limit damage done to data confidentiality*: Attacks on the confidentiality of data items can remain undetected, can be logged, can be monitored and the user alerted, can be prevented.

$$
\begin{aligned}
&\mathit{SecReqDataConf} == \mathit{GenericSecReq} \\
&\mathit{SecReqDataConfLevel} == \mathit{GenericSecReqLevel}
\end{aligned}
$$

3. *Limit damage done to code integrity*: Attacks on the integrity of executable code can remain undetected, can be logged, can be monitored and the user alerted, can be prevented.

$$
\begin{aligned}
&\mathit{SecReqCodeInt} == \mathit{GenericSecReq} \\
&\mathit{SecReqCodeIntLevel} == \mathit{GenericSecReqLevel}
\end{aligned}
$$

$$\mathit{SecurityRequirements} == \mathit{SecReqDataInt} \times \mathit{SecReqDataConf} \times \mathit{SecReqCodeInt}$$

This leaves us with a total of $4^3 = 64$ combinations of security requirements. When comparing two requirements pairs, we have $64 \times 64 = 4,096$ combinations. Of these 1,936 pairs are comparable ($=$, $>$, $<$) and 2,160 pairs are incomparable.

We do not expect this to be a significant problem. Security requirements are typically fixed for a product's use, so there is rather a lower bound on the requirements a product must comply with. Variations are expected to occur then mostly along one or two axes, or by the general addition of logging or alert capabilities.

6.3 Attacker capabilities

The metrics for necessary *attacker capabilities* also form a lattice by construction. A value of the metric is a four-tuple with its components taken from four inverse sequences, respectively. We recall the four axes from section 3.2 on page 48 and define the according sequences.

1. *Attack initiation capability*: control when an attack is begun and whether it can be attempted repeatedly.

$$
\begin{aligned}
&\mathit{AttCapInit} ::= \\
&\quad \mathit{InitByUser} \mid \mathit{InitAutomatically} \mid \mathit{InitByAttacker} \\
&\mathit{AttCapInitLevel} == \{ \\
&\quad (\mathit{InitByUser} \mapsto 1), \\
&\quad (\mathit{InitAutomatically} \mapsto 2), \\
&\quad (\mathit{InitByAttacker} \mapsto 3)\}
\end{aligned}
$$

2. *Available time to attack*: capability to attempt attack over time.

$$
\begin{aligned}
&\mathit{AttCapTime} ::= \\
&\quad \mathit{TimeOnce} \mid \mathit{TimeSeveral} \mid \mathit{TimeThroughout} \\
&\mathit{AttCapTimeLevel} == \{ \\
&\quad (\mathit{TimeOnce} \mapsto 1), \\
&\quad (\mathit{TimeSeveral} \mapsto 2), \\
&\quad (\mathit{TimeThroughout} \mapsto 3)\}
\end{aligned}
$$

3. *Attack variation*: adjust attack while in progress.

$$
\begin{aligned}
&\mathit{AttCapVariation} ::= \\
&\quad \mathit{CustomiseNot} \mid \mathit{CustomiseNoTuring} \mid \mathit{CustomiseTuring} \\
&\mathit{AttCapVariationLevel} == \{ \\
&\quad (\mathit{CustomiseNot} \mapsto 1), \\
&\quad (\mathit{CustomiseNoTuring} \mapsto 2), \\
&\quad (\mathit{CustomiseTuring} \mapsto 3)\}
\end{aligned}
$$

4. *Influence on user*: exercise influence on local human user.

Table 6.1: Necessary capabilities for attacks in repository

Generic Attack	Initiation	Time	Variation	User
Modify stored data item	3	1	1	1
Retrieve stored data item	3	1	1	1
Modify code in memory	3	1	1	1
Modify stored code module	3	1	1	1
Add stored code module	2	1	1	1
Modify reference to code	2	1	1	1
Initiate, send data	3	2	1	1
Respond, send data	2	2	2	1
Modify parameters	3	1	1	1
Modify reference to data	2	1	1	1
Simulate user input	3	2	1–3	1
Modify user interface	1	2–3	2	2–3
Modify data for decisions	3	1	1	1

$$\begin{aligned}
&\mathit{AttCapUser} ::= \\
&\quad \mathit{UserNot} \mid \mathit{UserOnce} \mid \mathit{UserRepeatedly} \\
&\mathit{AttCapUserLevel} == \{ \\
&\quad (\mathit{UserNot} \mapsto 1), \\
&\quad (\mathit{UserOnce} \mapsto 2), \\
&\quad (\mathit{UserRepeatedly} \mapsto 3)\}
\end{aligned}$$

The complete level of attacker capability is the four-tuple composed of values on the four base scales:

$$\mathit{AttackerCapabilities} == \mathit{AttCapInit} \times \mathit{AttCapTime} \times \mathit{AttCapVariation} \times \mathit{AttCapUser}$$

6.3.1 Necessary attacker capabilities

The capabilities needed by an attacker to perform an attack in the repository are shown in table 6.1. We require that an attacker is able to reliably perform an attack once its prerequisites are met. Probabilistic success, i.e., attacks that may or may not succeed, is not in the domain of attacker capabilities. We cover it by distinguishing between *undetected*, *detected*, and *immediately detected* attacks on our scale for security requirements.

Looking at table 6.1, we see that the attacks related to IPC and user interface are among the hardest to perform, and that the attacks involving references require least capabilities of an adversary.

Most often required capabilities are attack initiation by the adversary and being able to perform an attack at least once. Influence on the user is used least often. Because an architectural description is fixed, it is not surprising that generic attacks on an application's architecture do not need much customizing. And only where the local human user is involved, influence on the user is of relevance.

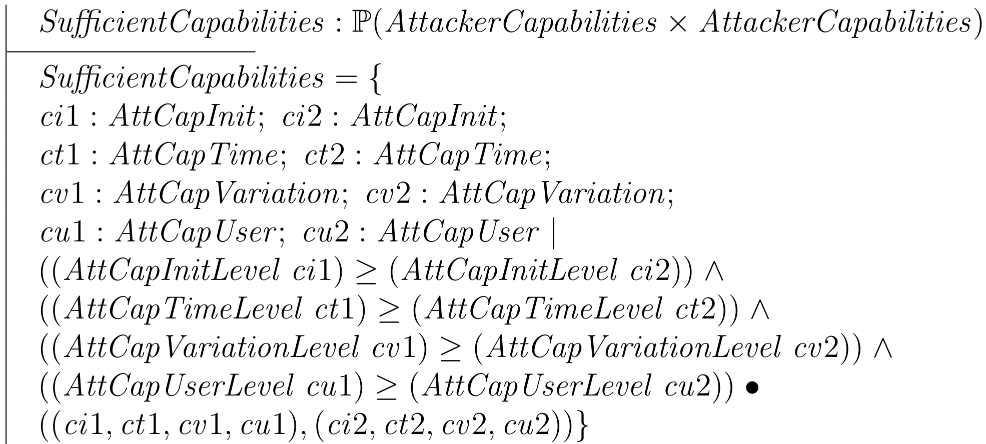
There are some combinations of values of the attacker capabilities metric for which no attack exists in the repository. This could be an indication that there are undiscovered attacks, e.g., attacks that are launched only based on user action and require several attempts and simulation of human behaviour (1–2–3–1).

6.4 Resistance classes

Finally, we combine security requirement levels, attacker capability levels and the attack repository as outlined in chapter 3. We determine to which resistance class a software product belongs, expressed by its architectural description.

An attacker is described by the capabilities possessed.

The relation *SufficientCapabilities* contains pairs of attacker capabilities that are comparable and of which the first is on a higher or equal level compared with the second.



For each of the 13 attacks in the attack repository we add a schema conditioning application of the attack upon the capabilities of an attacker needed to perform the attack. An overview of these capabilities is given in table 6.1.

Construction of the schemas is straightforward: *Attacker* and an *attack schema* are combined, and the predicate part is based on the necessary attacker capabilities that are included in the attack description in the repository and repeated in table 6.1. That way, an attack is only attempted if its prerequisites are met.

For the first attack, refer to the attack description on page 134.

AttemptAttackModifyStoredDataComponent
Attacker
AttackModifyStoredDataComponent

$$(capabilities, ((AttCapInitLevel\sim)3, (AttCapTimeLevel\sim)1, (AttCapVariationLevel\sim)1, (AttCapUserLevel\sim)1)) \in SufficientCapabilities$$

The following attack can be found in the attack repository on page 138.

AttemptAttackObserveStoredDataComponent
Attacker
AttackObserveStoredDataComponent

$$(capabilities, ((AttCapInitLevel\sim)3, (AttCapTimeLevel\sim)1, (AttCapVariationLevel\sim)1, (AttCapUserLevel\sim)1)) \in SufficientCapabilities$$

The following attack can be found in the attack repository on page 140.

AttemptAttackModifyCodeInMemory
Attacker
AttackModifyCodeInMemory

$$(capabilities, ((AttCapInitLevel\sim)3, (AttCapTimeLevel\sim)1, (AttCapVariationLevel\sim)1, (AttCapUserLevel\sim)1)) \in SufficientCapabilities$$

The following attack can be found in the attack repository on page 142.

AttemptAttackModifyStoredCodeModule
Attacker
AttackModifyStoredCodeModule

$$(capabilities, ((AttCapInitLevel\sim)3, (AttCapTimeLevel\sim)1, (AttCapVariationLevel\sim)1, (AttCapUserLevel\sim)1)) \in SufficientCapabilities$$

The following attack can be found in the attack repository on page 144.

AttemptAttackAddStoredCodeModule

Attacker
AttackAddStoredCodeModule

$(capabilities,$
$((AttCapInitLevel^\sim)2, (AttCapTimeLevel^\sim)1,$
$(AttCapVariationLevel^\sim)1, (AttCapUserLevel^\sim)1))$
$\in SufficientCapabilities$

The following attack can be found in the attack repository on page 147.

AttemptAttackModifyReferenceToStoredCodeModule

Attacker
AttackModifyReferenceToStoredCodeModule

$(capabilities,$
$((AttCapInitLevel^\sim)2, (AttCapTimeLevel^\sim)1,$
$(AttCapVariationLevel^\sim)1, (AttCapUserLevel^\sim)1))$
$\in SufficientCapabilities$

The following attack can be found in the attack repository on page 150.

AttemptAttackInitiateCommunicationAndSendData

Attacker
AttackInitiateCommunicationAndSendData

$(capabilities,$
$((AttCapInitLevel^\sim)3, (AttCapTimeLevel^\sim)2,$
$(AttCapVariationLevel^\sim)1, (AttCapUserLevel^\sim)1))$
$\in SufficientCapabilities$

The following attack can be found in the attack repository on page 153.

AttemptAttackRespondCommunicationAndSendData
Attacker
AttackRespondCommunicationAndSendData

$(capabilities,$
$((AttCapInitLevel\sim)2, (AttCapTimeLevel\sim)2,$
$(AttCapVariationLevel\sim)2, (AttCapUserLevel\sim)1))$
$\in SufficientCapabilities$

The following attack can be found in the attack repository on page 155.

AttemptAttackModifyStoredParameters
Attacker
AttackModifyStoredParameters

$(capabilities,$
$((AttCapInitLevel\sim)3, (AttCapTimeLevel\sim)1,$
$(AttCapVariationLevel\sim)1, (AttCapUserLevel\sim)1))$
$\in SufficientCapabilities$

The following attack can be found in the attack repository on page 157.

AttemptAttackModifyReferenceToStoredParameters
Attacker
AttackModifyReferenceToStoredParameters

$(capabilities,$
$((AttCapInitLevel\sim)2, (AttCapTimeLevel\sim)1,$
$(AttCapVariationLevel\sim)1, (AttCapUserLevel\sim)1))$
$\in SufficientCapabilities$

The following attack can be found in the attack repository on page 159.

$$
\begin{array}{l}
\textbf{AttemptAttackSimulateUserInput} \\
\hline
Attacker \\
AttackSimulateUserInput \\
\hline
((capabilities, \\
\quad ((AttCapInitLevel\sim)3, (AttCapTimeLevel\sim)2, \\
\quad (AttCapVariationLevel\sim)1, (AttCapUserLevel\sim)1)) \\
\quad \in SufficientCapabilities) \lor \\
((capabilities, \\
\quad ((AttCapInitLevel\sim)3, (AttCapTimeLevel\sim)2, \\
\quad (AttCapVariationLevel\sim)2, (AttCapUserLevel\sim)1)) \\
\quad \in SufficientCapabilities) \lor \\
((capabilities, \\
\quad ((AttCapInitLevel\sim)3, (AttCapTimeLevel\sim)2, \\
\quad (AttCapVariationLevel\sim)3, (AttCapUserLevel\sim)1)) \\
\quad \in SufficientCapabilities)
\end{array}
$$

The following attack can be found in the attack repository on page 161.

$$
\begin{array}{l}
\textbf{AttemptAttackModifyUserInterfaceObject} \\
\hline
Attacker \\
AttackModifyUserInterfaceObject \\
\hline
((capabilities, \\
\quad ((AttCapInitLevel\sim)1, (AttCapTimeLevel\sim)2, \\
\quad (AttCapVariationLevel\sim)2, (AttCapUserLevel\sim)2)) \\
\quad \in SufficientCapabilities) \lor \\
((capabilities, \\
\quad ((AttCapInitLevel\sim)1, (AttCapTimeLevel\sim)3, \\
\quad (AttCapVariationLevel\sim)2, (AttCapUserLevel\sim)2)) \\
\quad \in SufficientCapabilities) \lor \\
((capabilities, \\
\quad ((AttCapInitLevel\sim)1, (AttCapTimeLevel\sim)2, \\
\quad (AttCapVariationLevel\sim)2, (AttCapUserLevel\sim)3)) \\
\quad \in SufficientCapabilities) \lor \\
((capabilities, \\
\quad ((AttCapInitLevel\sim)1, (AttCapTimeLevel\sim)3, \\
\quad (AttCapVariationLevel\sim)2, (AttCapUserLevel\sim)3)) \\
\quad \in SufficientCapabilities)
\end{array}
$$

This last attack method can be found in the attack repository on page 163.

$$
\begin{array}{|l}
\multicolumn{1}{l}{__ \mathit{AttemptAttackModifyStoredDataComponentForDecisions} __} \\
\mathit{Attacker} \\
\mathit{AttackModifyStoredDataComponentForDecisions} \\
\hline
(\mathit{capabilities}, \\
\quad ((\mathit{AttCapInitLevel}^\sim)3, (\mathit{AttCapTimeLevel}^\sim)1, \\
\quad (\mathit{AttCapVariationLevel}^\sim)1, (\mathit{AttCapUserLevel}^\sim)1)) \\
\quad \in \mathit{SufficientCapabilities} \\
\hline
\end{array}
$$

The *AttemptAttacks* schema combines all conditioned attacks from the attack repository.

$$
\begin{array}{|l}
\multicolumn{1}{l}{__ \mathit{AttemptAttacks} __} \\
\mathit{AttemptAttackModifyStoredDataComponent} \\
\mathit{AttemptAttackObserveStoredDataComponent} \\
\mathit{AttemptAttackModifyCodeInMemory} \\
\mathit{AttemptAttackModifyStoredCodeModule} \\
\mathit{AttemptAttackAddStoredCodeModule} \\
\mathit{AttemptAttackModifyReferenceToStoredCodeModule} \\
\mathit{AttemptAttackInitiateCommunicationAndSendData} \\
\mathit{AttemptAttackRespondCommunicationAndSendData} \\
\mathit{AttemptAttackModifyStoredParameters} \\
\mathit{AttemptAttackModifyReferenceToStoredParameters} \\
\mathit{AttemptAttackSimulateUserInput} \\
\mathit{AttemptAttackModifyUserInterfaceObject} \\
\mathit{AttemptAttackModifyStoredDataComponentForDecisions} \\
\hline
\end{array}
$$

The *LoggedComponents* schema defines a set *accessLogged* of components to which accesses are logged.

$$
\begin{array}{|l}
\multicolumn{1}{l}{__ \mathit{LoggedComponents} __} \\
\mathit{Configuration} \\
\mathit{accessLogged} : \mathbb{P}\, \mathit{COMPNAME} \\
\hline
\mathit{accessLogged} = \{ \\
\quad \mathit{comp} : \mathit{COMPNAME};\ \mathit{conn} : \mathit{CONNNAME};\ \mathit{cport} : \mathit{PORT};\ \mathit{cprole} : \mathit{ROLE} \mid \\
\quad ((\mathit{conn}, \mathit{cprole}), (\mathit{comp}, \mathit{cport})) \in \mathit{attachment} \land \\
\quad \mathit{cprole} = \mathit{ROLOGDATASOURCE} \\
\quad \bullet\ \mathit{comp}\} \\
\hline
\end{array}
$$

The *MonitoredComponents* schema defines a set *accessMonitored* of components to which accesses are monitored.

$$
\begin{array}{l}
\textit{MonitoredComponents} \\
\hline
\textit{Configuration} \\
\textit{accessMonitored} : \mathbb{P}\ \textit{COMPNAME} \\
\hline
\textit{accessMonitored} = \{ \\
\quad \textit{comp} : \textit{COMPNAME};\ \textit{conn} : \textit{CONNNAME};\ \textit{cport} : \textit{PORT};\ \textit{cprole} : \textit{ROLE} \mid \\
\quad ((\textit{conn}, \textit{cprole}), (\textit{comp}, \textit{cport})) \in \textit{attachment} \land \\
\quad \textit{cprole} = \textit{ROMONITORSOURCE} \\
\quad \bullet\ \textit{comp}\}
\end{array}
$$

The *SecurityRequirementsRestriction* schema offers three technical relations to express an upper bound of components in a security requirements level.

$$
\begin{array}{l}
\textit{SecurityRequirementsRestriction} \\
\hline
\textit{MaxSecReqDataInt} : \textit{SecReqDataInt} \leftrightarrow \textit{SecurityRequirements} \\
\textit{MaxSecReqDataConf} : \textit{SecReqDataConf} \leftrightarrow \textit{SecurityRequirements} \\
\textit{MaxSecReqCodeInt} : \textit{SecReqCodeInt} \leftrightarrow \textit{SecurityRequirements} \\
\hline
\textit{MaxSecReqDataInt} = \{ \\
\quad \textit{srdi} : \textit{SecReqDataInt};\ \textit{srdc} : \textit{SecReqDataConf};\ \textit{srci} : \textit{SecReqCodeInt} \bullet \\
\quad \textit{srdi} \mapsto (\textit{srdi}, \textit{srdc}, \textit{srci})\} \\
\textit{MaxSecReqDataConf} = \{ \\
\quad \textit{srdi} : \textit{SecReqDataInt};\ \textit{srdc} : \textit{SecReqDataConf};\ \textit{srci} : \textit{SecReqCodeInt} \bullet \\
\quad \textit{srdc} \mapsto (\textit{srdi}, \textit{srdc}, \textit{srci})\} \\
\textit{MaxSecReqCodeInt} = \{ \\
\quad \textit{srdi} : \textit{SecReqDataInt};\ \textit{srdc} : \textit{SecReqDataConf};\ \textit{srci} : \textit{SecReqCodeInt} \bullet \\
\quad \textit{srci} \mapsto (\textit{srdi}, \textit{srdc}, \textit{srci})\}
\end{array}
$$

If no data components are modified, the highest data integrity level (4) is achievable. If some data components are modified and monitored, the next highest level (3) is possible. Detected modification leads to a lower level (2), and undetected modification of data components forces the lowest level for data integrity (1). This is expressed by the *EvaluateDataIntegrity* schema.

EvaluateDataIntegrity

$$
\begin{array}{l}
\mathit{Configuration} \\
\mathit{LoggedComponents} \\
\mathit{MonitoredComponents} \\
\mathit{SystemHistory} \\
\mathit{SecurityRequirementsRestriction} \\
\mathit{HighestSatisfiableSecurityRequirements} : \mathit{SecurityRequirements} \\
\hline
(((\mathit{SecReqDataIntLevel}^{\sim})4), \\
\quad \mathit{HighestSatisfiableSecurityRequirements}) \in \mathit{MaxSecReqDataInt} \\
\exists\, ct : \mathit{COMPTYPE};\ cn : \mathit{COMPNAME};\ ev : \mathit{EVENTACTION} \bullet \\
\quad (ct, cn, ev) \in \operatorname{ran} \mathit{history} \land \\
\qquad ((ev = \mathit{EVENT_ACCESS_MODIFY}) \lor \\
\qquad (ev = \mathit{EVENT_ACCESS_DELETE}) \lor \\
\qquad (ev = \mathit{EVENT_ACCESS_RENAME})) \land \\
\quad ct = \mathit{CTSUBJECTADVERSARY} \land \\
\quad (\mathit{components}\ cn).\mathit{owner} = \mathit{VICTIM} \land \\
\quad (\mathit{components}\ cn).\mathit{componentType} = \mathit{CTDATA} \land \\
\qquad (((\mathit{SecReqDataIntLevel}^{\sim})1), \mathit{HighestSatisfiableSecurityRequirements}) \\
\qquad\quad \in \mathit{MaxSecReqDataInt} \lor \\
\qquad ((cn \in \mathit{accessLogged}) \land (((\mathit{SecReqDataIntLevel}^{\sim})2), \\
\qquad\quad \mathit{HighestSatisfiableSecurityRequirements}) \\
\qquad\quad \in \mathit{MaxSecReqDataInt}) \lor \\
\qquad ((cn \in \mathit{accessMonitored}) \land (((\mathit{SecReqDataIntLevel}^{\sim})3), \\
\qquad\quad \mathit{HighestSatisfiableSecurityRequirements}) \\
\qquad\quad \in \mathit{MaxSecReqDataInt})
\end{array}
$$

If the contents of all data components are kept secret against an adversary, the highest data confidentiality level (4) is achievable. If content of some data components is revealed, but access is monitored, the next highest level (3) is possible. Detected observation leads to a lower level (2), and undetected violation of confidentiality of data components forces the lowest level for data confidentiality (1). This is expressed by the *EvaluateDataConfidentiality* schema.

$$
\begin{array}{l}
\textit{EvaluateDataConfidentiality} \\
\hline
\textit{Configuration} \\
\textit{LoggedComponents} \\
\textit{MonitoredComponents} \\
\textit{SystemHistory} \\
\textit{SecurityRequirementsRestriction} \\
\textit{HighestSatisfiableSecurityRequirements} : \textit{SecurityRequirements} \\
\hline
(((\textit{SecReqDataConfLevel}^{\sim})4), \\
\qquad \textit{HighestSatisfiableSecurityRequirements}) \in \textit{MaxSecReqDataConf} \\
\exists\, ct : \textit{COMPTYPE};\ cn : \textit{COMPNAME};\ ev : \textit{EVENTACTION} \bullet \\
\qquad (ct, cn, ev) \in \operatorname{ran} \textit{history} \land \\
\qquad ev = \textit{EVENT_ACCESS_OBSERVE} \land \\
\qquad ct = \textit{CTSUBJECTADVERSARY} \land \\
\qquad (\textit{components}\ cn).\textit{owner} = \textit{VICTIM} \land \\
\qquad (\textit{components}\ cn).\textit{componentType} = \textit{CTDATA} \land \\
\qquad\qquad (((\textit{SecReqDataConfLevel}^{\sim})1), \textit{HighestSatisfiableSecurityRequirements}) \\
\qquad\qquad\qquad \in \textit{MaxSecReqDataConf} \lor \\
\qquad\qquad ((cn \in \textit{accessLogged}) \land (((\textit{SecReqDataConfLevel}^{\sim})2), \\
\qquad\qquad\qquad \textit{HighestSatisfiableSecurityRequirements}) \\
\qquad\qquad\qquad \in \textit{MaxSecReqDataConf}) \lor \\
\qquad\qquad ((cn \in \textit{accessMonitored}) \land (((\textit{SecReqDataConfLevel}^{\sim})3), \\
\qquad\qquad\qquad \textit{HighestSatisfiableSecurityRequirements}) \\
\qquad\qquad\qquad \in \textit{MaxSecReqDataConf}) \\
\hline
\end{array}
$$

Code integrity is similar to data integrity. In addition, some *invoke* operations must be checked.

$$
\begin{array}{l}
\textit{EvaluateCodeIntegrity} \\
\hline
\textit{Configuration} \\
\textit{LoggedComponents} \\
\textit{MonitoredComponents} \\
\textit{SystemHistory} \\
\textit{SecurityRequirementsRestriction} \\
\textit{HighestSatisfiableSecurityRequirements} : \textit{SecurityRequirements} \\
\hline
(((\textit{SecReqCodeIntLevel}^{\sim})4), \\
\quad \textit{HighestSatisfiableSecurityRequirements}) \in \textit{MaxSecReqCodeInt} \\
\exists\, ct : \textit{COMPTYPE};\ cn : \textit{COMPNAME};\ ev : \textit{EVENTACTION} \bullet \\
\quad (ct, cn, ev) \in \operatorname{ran} \textit{history} \land \\
\quad ((ct = \textit{CTSUBJECTADVERSARY} \land \\
\quad\quad ((ev = \textit{EVENT_ACCESS_MODIFY}) \lor \\
\quad\quad (ev = \textit{EVENT_ACCESS_DELETE}) \lor \\
\quad\quad (ev = \textit{EVENT_ACCESS_RENAME})) \land \\
\quad\quad (\textit{components}\ cn).\textit{owner} = \textit{VICTIM}) \lor \\
\quad (ct = \textit{CTSUBJECTVICTIM} \land \\
\quad\quad ev = \textit{EVENT_ACCESS_INVOKE} \land \\
\quad\quad (\textit{components}\ cn).\textit{owner} = \textit{ADVERSARY})) \land \\
\quad (\textit{components}\ cn).\textit{componentType} = \textit{CTDATA} \land \\
\quad\quad (((\textit{SecReqCodeIntLevel}^{\sim})1), \textit{HighestSatisfiableSecurityRequirements}) \\
\quad\quad\quad \in \textit{MaxSecReqCodeInt} \lor \\
\quad\quad ((cn \in \textit{accessLogged}) \land (((\textit{SecReqCodeIntLevel}^{\sim})2), \\
\quad\quad\quad \textit{HighestSatisfiableSecurityRequirements}) \\
\quad\quad\quad \in \textit{MaxSecReqCodeInt}) \lor \\
\quad\quad ((cn \in \textit{accessMonitored}) \land (((\textit{SecReqCodeIntLevel}^{\sim})3), \\
\quad\quad\quad \textit{HighestSatisfiableSecurityRequirements}) \\
\quad\quad\quad \in \textit{MaxSecReqCodeInt})
\end{array}
$$

The relation *SatisfyingSecurityRequirements* contains pairs of security requirement levels that are comparable and of which the first is on a higher or equal level compared with the second.

$$
\begin{array}{l}
\textit{SatisfyingSecurityRequirements} : \mathbb{P}(\textit{SecurityRequirements} \times \textit{SecurityRequirements}) \\
\hline
\textit{SatisfyingSecurityRequirements} = \{ \\
di1 : \textit{SecReqDataInt};\ di2 : \textit{SecReqDataInt}; \\
dc1 : \textit{SecReqDataConf};\ dc2 : \textit{SecReqDataConf}; \\
ci1 : \textit{SecReqCodeInt};\ ci2 : \textit{SecReqCodeInt} \mid \\
((\textit{SecReqDataIntLevel}\ di1) \geq (\textit{SecReqDataIntLevel}\ di2)) \land \\
((\textit{SecReqDataConfLevel}\ dc1) \geq (\textit{SecReqDataConfLevel}\ dc2)) \land \\
((\textit{SecReqCodeIntLevel}\ ci1) \geq (\textit{SecReqCodeIntLevel}\ ci2)) \bullet \\
((di1, dc1, ci1), (di2, dc2, ci2)) \\
\}
\end{array}
$$

The *SECURITYSTATUS* of an evaluated architectural description/configuration in-

dicates whether it fulfils the desired security requirements if faced with an adversary possessing the accepted capabilities.

$$SECURITYSTATUS ::= SecureEnough \mid NotSecureEnough$$

With the *EvaluateSystemState* schema we check an architectural description against a desired security requirements level. The *ProductSecurityStatus* variable provides the result of the check.

```
┌─ EvaluateSystemState ─────────────────────────────
│ EvaluateDataIntegrity
│ EvaluateDataConfidentiality
│ EvaluateCodeIntegrity
│ DesiredSecurityRequirements : SecurityRequirements
│ ProductSecurityStatus : SECURITYSTATUS
├───────────────
│ (HighestSatisfiableSecurityRequirements, DesiredSecurityRequirements)
│     ∈ SatisfyingSecurityRequirements ⇔
│ ProductSecurityStatus = SecureEnough
└───────────────────────────────────────────────────
```

Determination of the membership of a software product in a resistance class is then done by a post-hoc evaluation of the system state after all possible simulated attacks on the architecture of a program.

$$\begin{aligned}&EvaluateAttackedSystem == \\ &\quad (((Configuration \land InitialStateCorrespondence \land \\ &\quad Attacker \land AttemptAttacks) \upharpoonright \\ &\quad (Configuration \land \Delta SystemHistory)) \fatsemi \\ &\qquad EvaluateSystemState)\end{aligned}$$

The level of resistance is reported by the *ResistanceClass* variable and consists of an attacker capability level and a security requirements level.

```
┌─ DetermineResistanceClass ────────────────────────
│ Attacker
│ EvaluateAttackedSystem
│ ResistanceClass : AttackerCapabilities × SecurityRequirements
├───────────────
│ ResistanceClass = (capabilities, DesiredSecurityRequirements)
│ ProductSecurityStatus = SecureEnough
└───────────────────────────────────────────────────
```

Chapter 7

Analysis of architectural differences and their influence on effectiveness of counter-measures

Chapter summary: In this chapter we show how an architectural description of a program can be analysed with our methods. Our example is a homebanking application. We compare two versions of the product: both are in the same resistance class, but our metrics reveal a negative trend that could lead to further vulnerability in the future. Based on our results, we propose improvements in the software architecture.

7.1 Architectural changes

An architecture is a system's implementation-independent structure given by its components and the relations among them and to the environment. Architectural weaknesses are weaknesses in the design of a system (e.g., missing/weak authentication, bypassing of controls, poor choice of parameters). They are a mismatch between security requirements and system specification and exist regardless of the quality of an architecture's implementation. Attacks on a system's implementation-independent architecture may be harder to detect and harder to correct by simple means after deployment, compared with errors in an implementation.

Change requests for functional requirements can lead to a change in system design. This in turn can influence whether the system still complies with the security requirements. In the following sections we compare the resistance classes of the software architecture of different versions of the same product.

7.2 Homebanking with FinTS/HBCI

Owing to a recent rise of social engineering – *phishing* – attacks on internet banking systems, homebanking applications are pushed as a means to combat this threat. Transactions can be prepared at a local machine where they are signed and then transmitted encrypted to the bank's server. No transaction numbers are involved, and hence, data for authorization of transactions cannot be captured and used by an adversary. A smart card issued by the bank and employed by the local user contains cryptographic keys to authenticate users and their transactions to the bank. Electronic signatures are computed

inside the card, and transactions are encrypted so that the bank is able to establish their authenticity, integrity, and their not being known to an adversary.

Access to the smart card is regulated by a personal identification number (PIN). It can be input locally to be relayed by the banking software to the card reader and the card. In many systems, locally input data is also available to other processes on the machine. A malicious process could thus eavesdrop on the input and retrieve the PIN for the card. As a standard counter measure the use of *class 2* card readers is recommended. These integrate a PIN pad and are capable of transmitting a PIN input at that PIN pad directly to the card. Since the PIN is never transmitted from the card reader to the PC, it cannot be obtained by other processes.

In this scenario, the existence of a local malicious process is hence assumed. In addition to eavesdropping on local input, that process has more capabilities for trying to influence the execution of the homebanking application (cf. sections 3.3 and 6.1 and also [SCL02], [Lan06b]).

The situation is comparable to the creation of electronic signatures by way of a smart card. In the homebanking case, the data to be signed is simpler and is always prepared by the application accessing the card. The data is only used for banking transactions and not for legally binding general purpose electronic signatures. The receiver of the data is the bank that in most cases also issues the card and recommends the banking software. The standard used by most German banks for their homebanking is FinTS/HBCI. [HBC00, Fin04]

As there are no publicly known cases outside the laboratory to date where there have been manipulations of bank transactions when using HBCI, banks tend to put responsibility for correct creation of transactions on the customer. As an example, we evaluate the homebanking application *StarMoney 5.0* by *Star Finanz – Software Entwicklung und Vertriebs GmbH*. They claim to have a market share of ca. 50% of the German market at present. A comparison with products of other manufacturers can be found in [LS07].

We expose the software structure based on analysis without knowledge of the source code or internal design documentation.

7.2.1 High-level architecture

Architecturally, the homebanking application with a total size of ca. `50 MB` consists of 2 small executable modules, `StartStarMoney.exe [60 KB]` for preparing the start of the main executable `StarMoney.exe [172 KB]`. (6 additional executable modules perform peripheral functionality: Configuring smart card readers attached to the system (`SCRSetup.exe`), conversion of older versions' data (`smkonv.exe`), T-Online Classic access (`CLGate32.exe`, `sfkclgateslave.exe`, `sfktonac.exe`), remote support (`NetViewer.exe`))

The executable modules are supported by 89 executable library files that extend the functionality of the main executable. They can be partitioned into 6 function groups: *Core banking* (18 libraries), *Smart card communication* (4 libraries), *Database access* (11 libraries), *Communication* (17 libraries), *User interface* (24 libraries), *Miscellaneous* (15 libraries). We list size, name, and purpose of each identified module and state the dependency relations among the executable modules and with data files in appendix B.

Execution dependency graphs for `StarMoney.exe` and `StartStarMoney.exe` are shown in figures 7-1 through 7-4 and 7-5. These two programs are used most often compared to the others and our analysis will henceforth focus on the two.

For enhanced clarity of the presentation, only data and executable components are included that stand for file with a `DLL`, `EXE`, or `INI` extension. Database and temporary files are left out as well as user interface components. These are nevertheless taken into account in our later metrics calculations.

Not shown in the figures are operating system libraries that are loaded before the invocation of StarMoney. When they are referenced incompletely, libraries with the same base name may be mapped into the address space of the new process. Nevertheless, a substitution of dynamically linked libraries does not have an effect in this case, because the references are resolved to the libraries already in memory.

A component is shown as a *rectangle* with a component name and a location (a related container) and connected to other components by lines. *Simple arrows* point from a host component to an executable component executed in the address space of the host. *Double arrows* point from an executable component to another component executed as a separated process. A *line* between an executable (`DLL`, `EXE`) and a non-executable (`INI`) component represents a data transfer connector. The *circle* end denotes the data component and the *filled circle* end denotes the executable.

The execution connectors are organized in a graph. It is acyclic which means that there are no cyclic dependencies of executable components on each other. Some components are referenced by many others, making them attractive targets, because code of these components will be executed with a high probability. Components also depend on many other executable components, either directly or indirectly. Hence, there are many possible vectors to introduce code in their address space.

In the upper right hand corner of the rectangle of an executable component we find its location. In our example case there are three groups of locations: operating system folders (`%WINDIR%`), application folders (`%INSTDIR%`), third party folder (`?`). Each location has the same access control configuration for the components it contains, but access control may vary across location groups. Operating system locations are well protected, third party locations are not under control, application locations are in the manufacturer's domain.

7.2.2 Metrics

With respect to StarMoney's architecture, our software security metrics from chapter 3 have the values shown in table 7.1. Measurements are performed by counting components and connectors of specific types and then relating the counts to each other. The formulae used are documented in the metrics definitions in chapter 3.

For metrics M_4, M_5, M_{10}, M_{11} values could not be obtained for all three programs, owing to a lack of detailed design documentation. It might be possible to reconstruct these values by spending more effort on advanced architecture discovery methods, e.g., reverse engineering, but this is left to future work. For `StarMoney.exe`, the values are complete. Upon availability of the successor version *6.0* examination was conducted to find out about changes in the program's architecture and changes in the corresponding metrics. We compare results in table 7.2.

StarMoney receives perfect scores for metrics M_9, M_{14}, and almost perfect values for metrics M_3 and M_6. Interestingly, the latter two metrics indicate a negative trend when comparing version 5.0 and 6.0. M_3 (Percentage of protected executables) goes down from 97% to 30%, and M_6 (Conformity of access permissions) goes down from 97% to 67%. Very low values are reported for 6 of 14 metrics for both versions.

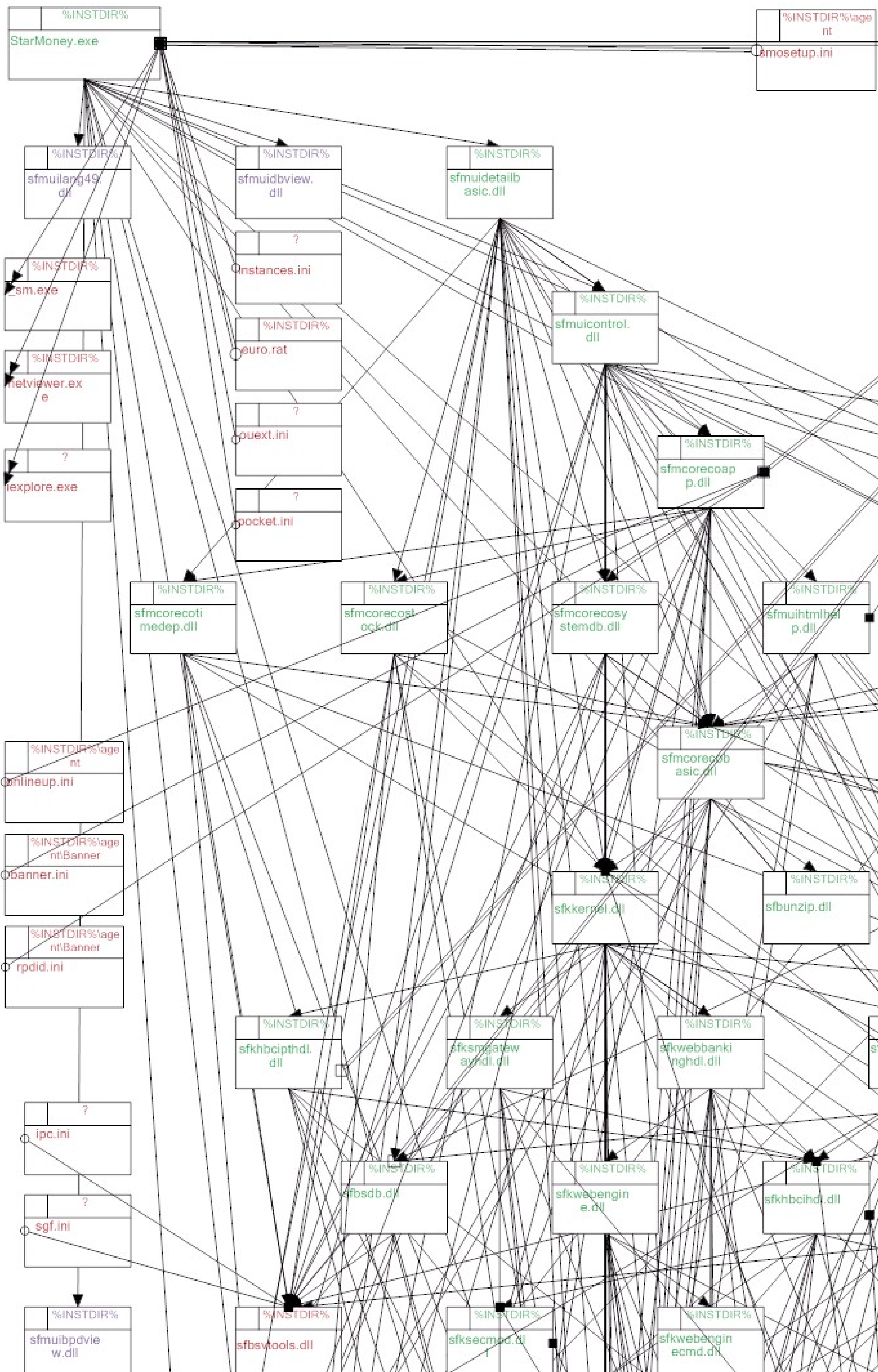


Figure 7-1: Dependency graph for executable modules used by `StarMoney.exe`, part (i)

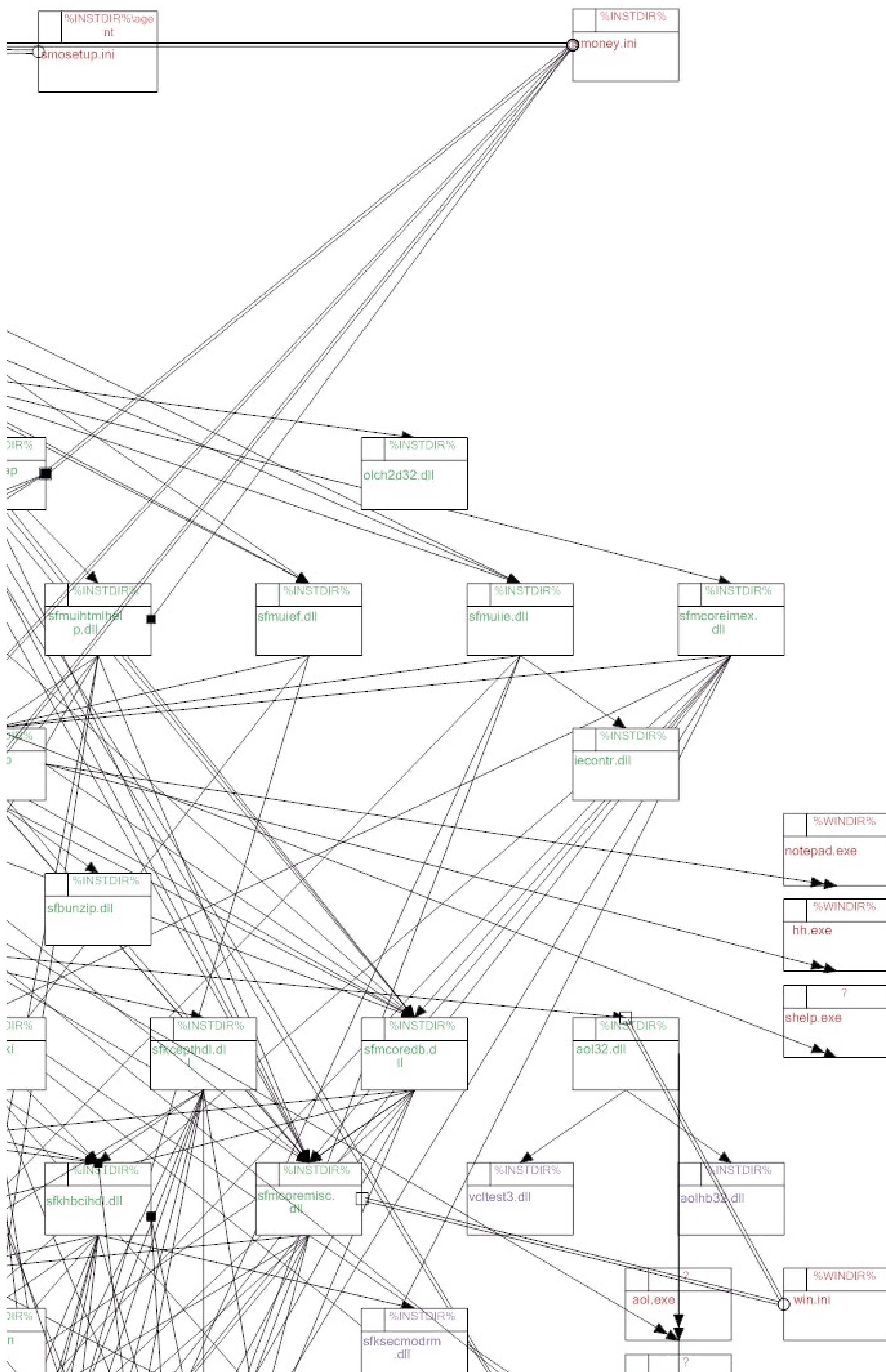


Figure 7-2: Dependency graph for executable modules used by `StarMoney.exe`, part (ii)

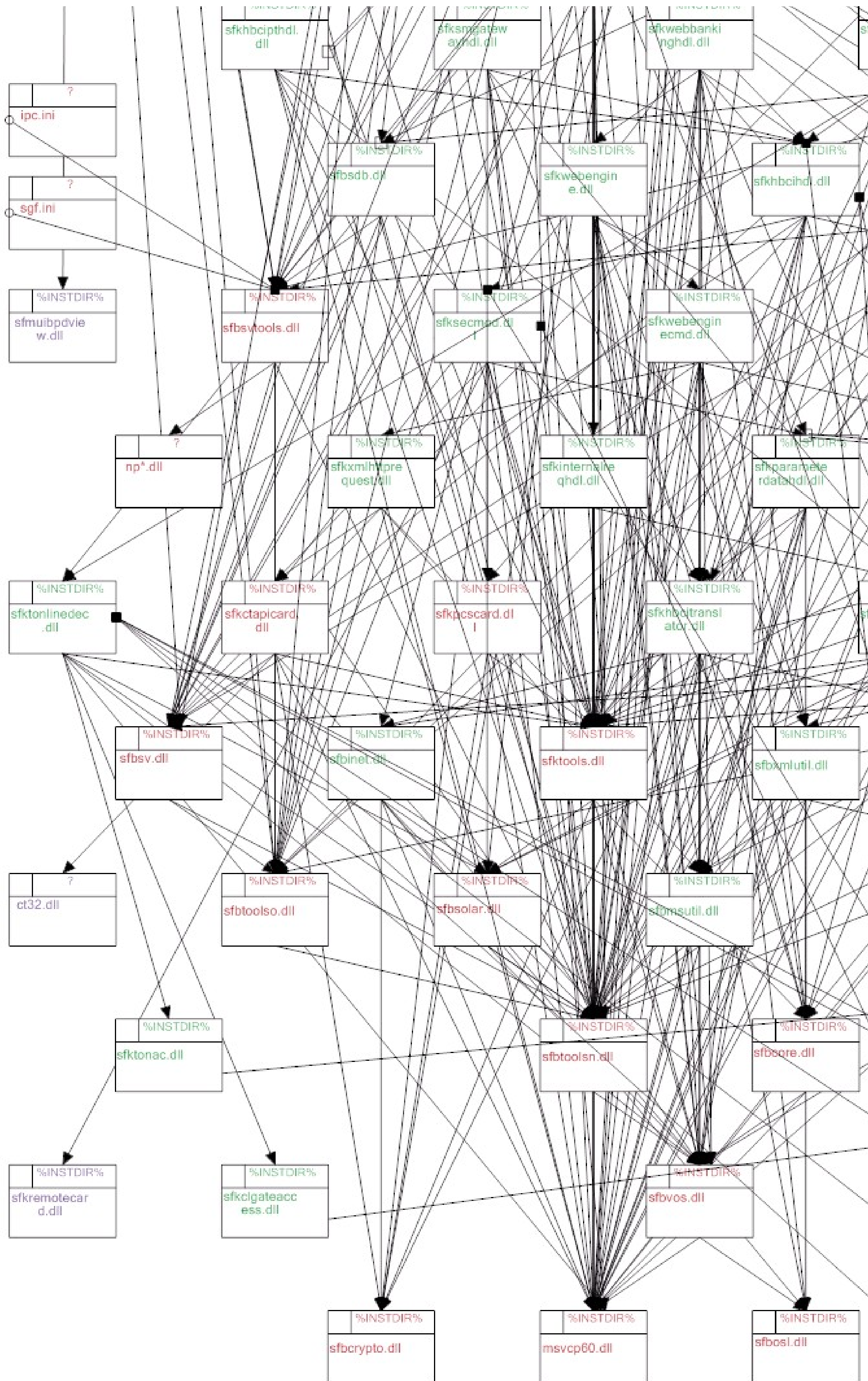


Figure 7-3: Dependency graph for executable modules used by `StarMoney.exe`, part (iii)

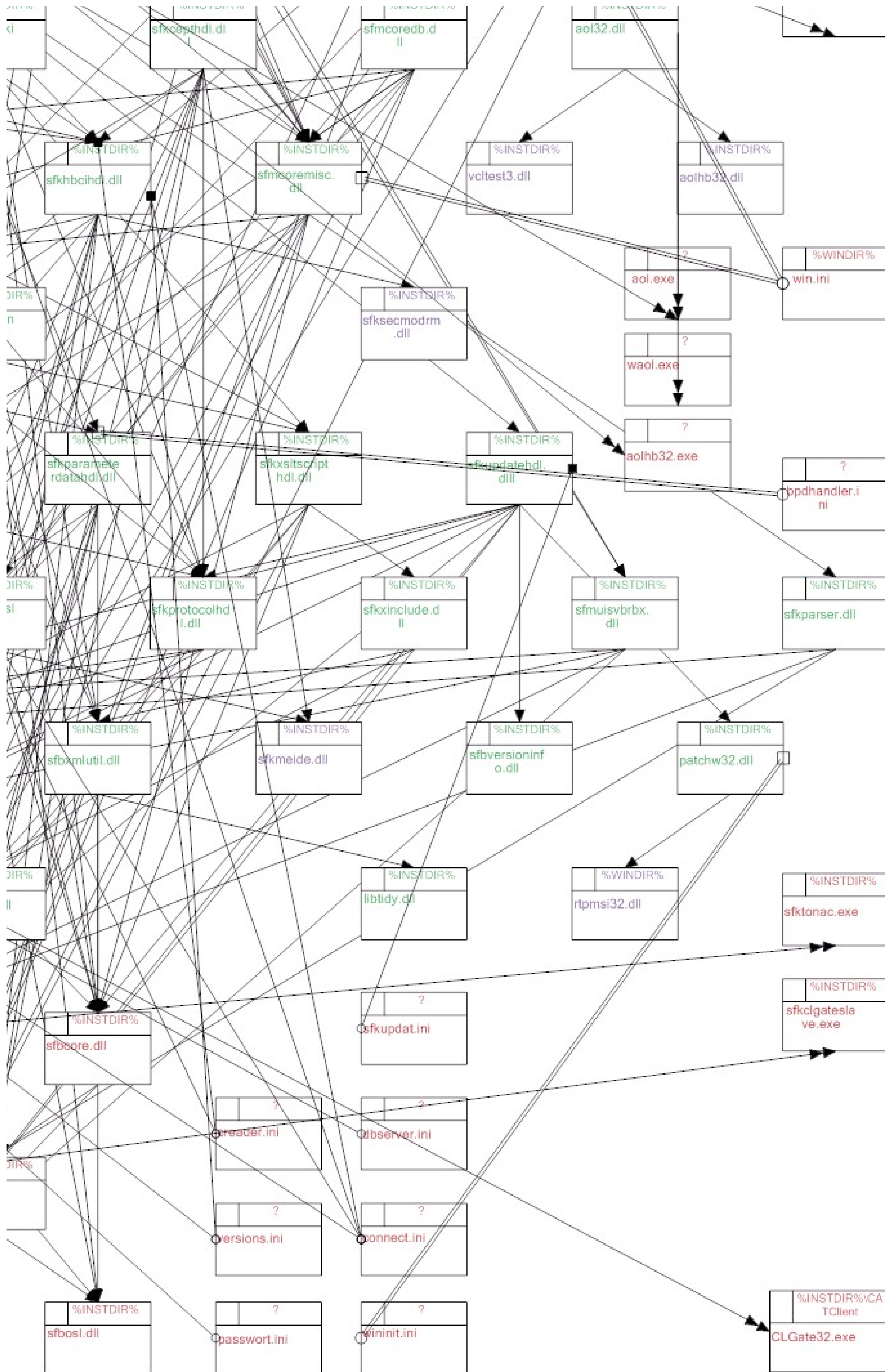


Figure 7-4: Dependency graph for executable modules used by `StarMoney.exe`, part (iv)

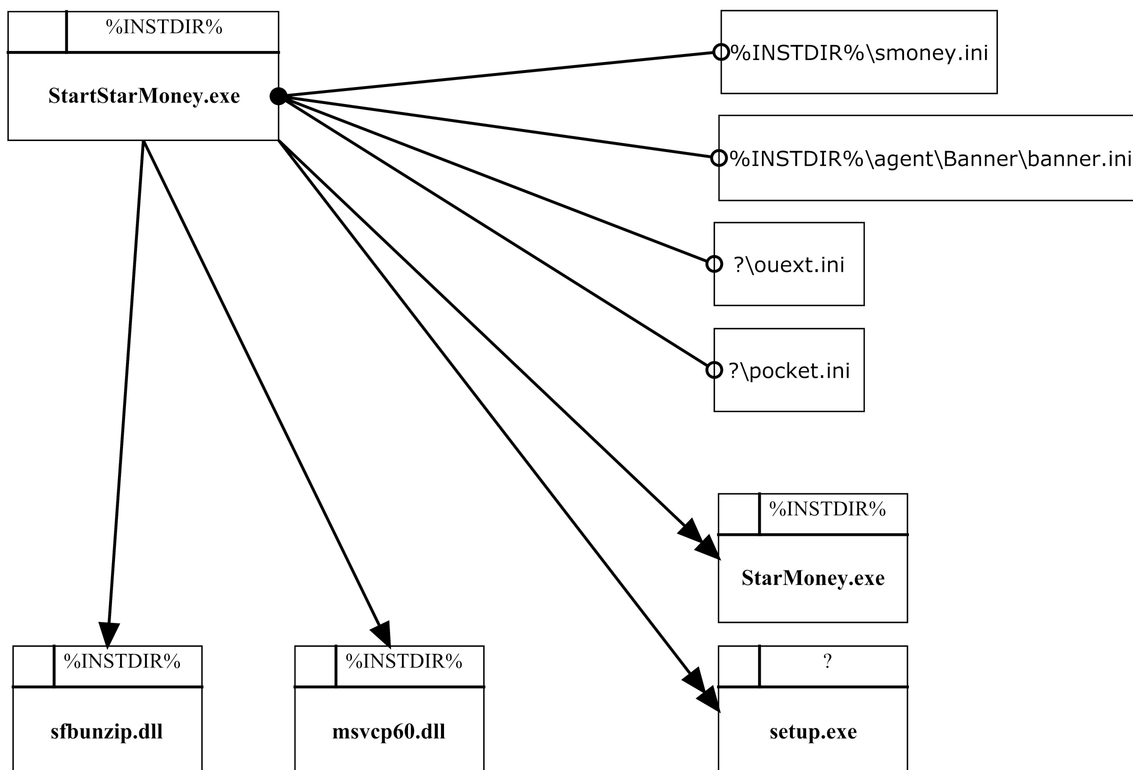


Figure 7-5: Executable data components and execute connectors of `StartStarMoney.exe`

Closer inspection reveals areas of concern. Intermediate data is not protected during operation, and the user interface lacks any protection against attacks on its integrity and confidentiality. There are many executable components that have a large number of dependencies among them.

Reason for the negative trend observed with respect to the latest product version is the removal of protection on installed application components by default and the introduction of more unprotected references to data components containing parameters. The application's architecture as a whole has not been changed much.

The single most important metric useful as an indicator is M_3 (Percentage of protected executables). Component coupling and management of connections is covered by M_2, M_5, M_8, M_{10}, M_{11}, M_{12}, and is the second most important indicator. If an application scores badly on these metrics, one does not need to investigate further metrics, because the unprotected attack surface provides ample opportunity for an attacker.

Metrics M_{13} and M_{14} do not seem to add information to the picture. This owes to our theoretical model where we do have only limited support of subject privileges and it owes to our case study with only a single process being executed to run the application.

Metric M_6 is a good indicator for the deployment planning phase and for installed products. It can reveal deviations in protection levels. These are not problems *per se*, but they require heightened management attention.

7.2.3 Resistance class

The resistance class of StarMoney 5.0 is as follows:

$$
\begin{array}{l}
(\\
\quad (\\
\qquad AttCapInit : InitByAttacker,\\
\qquad AttCapTime : TimeThroughout,\\
\qquad AttCapVariation : CustomiseNot,\\
\qquad AttCapUser : UserOnce\\
\quad),\\
\quad (\\
\qquad SecReqDataInt : Alert,\\
\qquad SecReqDataConf : Undetected,\\
\qquad SecReqCodeInt : Prevent\\
\quad)\\
)
\end{array}
$$

This means that violations of data integrity can be detected during operation while data confidentiality is not guaranteed and code integrity is only achieved if third-party components are adequately protected. The security requirements are met in the presence of an adversarial process on the target machine that is able to operate multiple times and at its own discretion. The attacking process must not be able to adapt the attack like a human adversary, and the attacker does not need to be able to influence the user more than once. These capabilities can typically be assumed in malware attacks on homebanking applications.

Formally, the resistance class is determined by proving that there is a model instance for the given configuration of the product and the *DesiredSecurityRequirements* schema variable used in conjunction with the *DetermineResistanceClass* schema. That proof is not included in our case study and left to future work.

The negative trend in metrics M_3, M_6, and M_8 indicates that future versions of StarMoney might only be members of lower resistance classes, e.g., by dropping from *Prevent* to *Undetected* for the *Code integrity* part.

7.2.4 Tool support

While architectural descriptions of small systems can be evaluated by hand, this can take a long time and become practically infeasible with large systems. Here, support by (automated) tools can reduce time spent on evaluation.

We surveyed several such tools to support the use of Z. Criteria in the initial selection process were functionality – we consider a type checker, typesetting support, and (interactive) proof facilities to be necessary –, availability of documentation and maintenance, and price. Seven programs were examined in the process:

- CADi$\mathbb{Z}$ is a collection of programs around Z, among them a *type checker*, typesetting macros in connection with LaTeX, an interactive browser for typeset and typechecked specifications, interactive proof support with its own language of proof tactics. It can be used with Linux or Windows NT with Cygwin. CADi$\mathbb{Z}$ is maintained and distributed freely by Ian Toyn of the University of York, UK.

- *Proof Power* is a collection of programs to support *specification* and *proof* with the*Isabelle/HOL* tool and in *Z* notation. It can be used with Linux or MacOS X. It can be obtained free of charge for academic use from Lemma 1 Ltd. It is industrial strength software with a user community and maintenance by a for-profit organization.

- *MOBY/OZ* is a graphical *editor* to construct *Z* specifications. It can be used with Linux and is freely distributed by the University of Oldenburg. Maintenance depends on the PhD students currently available and familiar with the tool. Correspondence suggested that use of LaTeX and a *Z* symbol translation chart would do almost equally well.

- *ZTC* is a *type checker* for *Z*. It can be used with Windows NT or Linux. The program is developed and distributed freely by DePaul University.

- *ROZ* is a program to *translate annotated UML* class diagrams to *Z* specifications. It can be used with Windows NT or Linux and is available from LSR-IMAG Grenoble.

- *Z Browser* is able to *present Z* specifications, especially for teaching. It can be used with Windows NT. ORA Canada distributes, but not maintains the product. Academic licences are available on an individual basis, otherwise there is a 14 days free evaluation period. There are unclear legal issues as to what usage restrictions apply.

- *Zeus-Z* is an *editor* and *syntax checker* for *Z*. It needs FrameMaker and Z/EVES. Software is available free of charge by ORA Canada, alas their homepage notes: "Distribution by ORA ceased as of 2005-06-03".

An overview of the tools and their functionality is given in table 7.3. A cross in a column indicates that the respective product supports the respective functionality denoted by the column heading.

Editor is the capability of editing *Z* specifications without writing typesetting source code (e.g., LaTeX). *Browser* refers to the ability of a tool to display a specification and allow navigation depending on variable definitions and types. *Type checker* is a tool's capability of checking a *Z* specifition for compliance with *Z*'s type model and verifying whether a given specification is syntactically correct. *Proof support* states if there is software support for (semi-)automatically proving properties of a specification. *Specials* is a column for significant functionality not fitting into one of the previous columns.

As a result of our examination we settle with CADiℤ for type-checking and browsing our specification. It is also used for proof support. Proof Power might offer industrial strength proof support, but CADiℤ allows to reuse our typeset specifications and integrates better with LaTeX. This spares us the preparation of separate input files for processing with our support tool.

MOBY/OZ does not offer significant help as soon as one is familiar with the *Z* LaTeXmacros and CADiℤ. Both CADiℤ and *Proof Power* are supported by user groups and can be used freely. *ROZ* offers too much "syntactic sugar", *Z Browser* and *Zeus-Z* do not offer stable distribution and licensing.

7.2.5 Possible architectural improvements

Based on our findings we have some suggestions for architectural improvements to yield a higher resistance class for future StarMoney.

Higher data integrity could be achieved by greater coverage of integrity verification mechanisms for data transfer. This is measured by M_4, M_5, and M_8.

Higher code integrity could be achieved by greater coverage and more uniformity of access permissions for executable components. This is measured by M_1, M_3, and M_6.

Higher requirements for attack variability could be achieved by limiting simulatability of user actions. This is measured by M_7, M_9, and M_{12}.

Higher requirements for ability to influence the user could be achieved by higher integrity for output displayed to the user on which the user makes decisions. This is measured by M_7 and M_{12}.

These recommendations are based on the technical vulnerability analysis of the product. They do not take into account non-technical measures, e.g., law enforcement. In addition, a rational adversary might prefer alternate approaches to attack homebanking accounts that are easier or cost less to perform at present, e.g., setting up a fake web server and directing clients to it.

7.3 Discussion

The effort to measure a product's security status proved to be reasonable. After collecting the raw data, the actual counting was a matter of hours by hand. A lot more time was spent to gather the raw data. In an ideal world, every product was accompanied by standardised machine-readable design documentation. Component types and attributes, call graphs and deployment diagrams could then be processed automatically. In our case study we had to proceed without source code or internal design documentation for the product. This required several days of skilled scrutiny and is unlikely to be feasible for industrial use.

Our 14 software security metrics were developed with design principles in mind. We can claim that all relevant aspects (expressed by the security requirements) are covered by the metrics. A correspondence argument for metrics and security requirements is also shown in table 3.18. If one were to build a relevancy-tree to decide which metrics to apply first and when to stop based on enough measurements, we suggest M_3 as the root node. Also high in the tree would be M_2, M_5, M_6, M_8, M_{10}, M_{11}, and M_{12}, while M_{13} and M_{14} would be on lower branches.

In our example we could have arrived at the conclusion that there are serious architectural vulnerabilities in *StarMoney* by evaluating M_3, M_6, M_{12}, and M_8 alone.

All metrics are normalized to a 0–100 scale. However, measurement results should not be combined simply by using arithmetical operations on these values. An aggregated value does not have any semantics here. It is possible to detect trends among product versions, and it is possible to distinguish values *0*, *100*, and *below 100*.

Validity of software security metrics in general is another recurring theme. Each metric can be gamed, i.e., some developers tend to adapt their products so that they get a good score on the software metrics while neglecting other design principles. With our approach to metrics development as explained in chapter 3 we have constructively embedded secure design principles in the metrics. A relevancy-tree involving an incomplete set of our 14 software security metrics might be susceptible to validation challenges, and this is left as an open question to future work.

Table 7.1: Values for security metrics applied to StarMoney 5.0

Metric	StarMoney.exe	StartStarMoney.exe	SCRSetup.exe
M_1: Centralisation of executable distribution sources	50%	50%	50%
M_2: Restriction of number of executable components	1% [1/(66+25+3)]	9% [1/(3+8+0)]	3% [1/(15+15+3)]
M_3: Percentage of protected executables	97% ($\frac{66+25}{66+25+3}$)	73% (8/11)	45% (15/33)
M_4: Percentage of protected intermediate storage components	0%	n/a	n/a
M_5: Percentage of access control instrumentation	28% [7/(16+9)]	n/a	n/a
M_6: Conformity of access permissions	97%	100%	91%
M_7: Percentage of logged invocations	0%	0%	0%
M_8: Percentage of authenticity/integrity preserving connectors	16% [7/(16+27)]	0%	0%
M_9: Percentage of unlogged security parameters	100%	100%	100%
M_{10}: Restriction of number of components with shared responsibility (server)	3% (1/37)	n/a	n/a
M_{11}: Restriction of number of components with multiple executable extensions	2% (1/49)	n/a	n/a
M_{12}: Percentage of trusted path connectors	0%	0%	0%
M_{13}: Restriction of number of privileges	25%	25%	25%
M_{14}: Restriction of number of processes sharing a privilege	100%	100%	100%

Table 7.2: Comparison of metrics values for StarMoney 5.0 and 6.0

Metric	`StarMoney.exe` 5.0	`StarMoney.exe` 6.0
M_1: Centralisation of executable distribution sources	50%	50%
M_2: Restriction of number of executable components	1% [1/(66+25+3)]	1% [1/(66+28+3)]
M_3: Percentage of protected executables	97% ($\frac{66+25}{66+25+3}$)	30% ($\frac{28+1}{66+28+3}$)
M_4: Percentage of protected intermediate storage components	0%	0%
M_5: Percentage of access control instrumentation	28% [7/(16+9)]	28% [7/(16+9)]
M_6: Conformity of access permissions	97%	67%
M_7: Percentage of logged invocations	0%	0%
M_8: Percentage of authenticity/integrity preserving connectors	16% [7/(16+27)]	15% [7/(16+30)]
M_9: Percentage of unlogged security parameters	100%	100%
M_{10}: Restriction of number of components with shared responsibility (server)	3% (1/37)	3% (1/37)
M_{11}: Restriction of number of components with multiple executable extensions	2% (1/49)	2% (1/49)
M_{12}: Percentage of trusted path connectors	0%	0%
M_{13}: Restriction of number of privileges	25%	25%
M_{14}: Restriction of number of processes sharing a privilege	100%	100%

	Editor	*Browser*	*Type checker*	*Proof support*	*Specials*
CADi$\mathbb{Z}$	(x)	X	X	X	-
Proof Power	-	-	-	X	-
MOBY/OZ	X	-	-	-	-
ZTC	-	-	X	-	-
ROZ	-	-	-	-	Convert UML
Z Browser	-	X	-	-	-
Zeus-Z	X	-	-	-	-

Table 7.3: Tool support for *Z*

Chapter 8

Conclusions

Not everything that can be counted counts,
and not everything that counts can be counted.
— Frequently attributed to Albert Einstein

Software security metrics are to a large extent still an uncharted territory, elevated by its having been named among the *grand research challenges* in information systems by the Computing Research Association in 2003. [Ass03]

We have shown how security requirements and attacker capabilities for local malware attacks can be formally described. Our extended model of a computer system is based on the Common Criteria for Information Technology Security Evaluation (CC). With it we are able to describe and analyse attacks of local malicious processes on fundamental software design vulnerabilities. A repository of generic attacks helps to check whether some given systems lie in the same resistance class. The lattice order on resistance classes allows comparison of similar systems. Our model is flexible enough to incorporate future attack methods that are not known today.

8.1 Contributions

This dissertation has four main contributions: A generic model of an operating system from a security perspective, a repository of typical attack methods, a set of resistance classes, and an identification of software architecture metrics pertaining to ordered security levels.

Attacks on a software's architecture have been little explored previously. We look at the relationship between software structure, attacks, and resulting resilience of software. A *generic model of an operating system*, compatible with the established Common Criteria for information technology security evaluation [CC299b], allows the expression of software architecture from a security perspective. The software in question, adversarial processes, and their capabilities can be analysed using a single consistent syntax and semantics. Such a generic model has not been found in the open literature. Alas, it might have been developed in secret by government or industrial bodies involved with security assessment. Evaluation on the architectural level can be done more easily than evaluating the complete source code of a project. It could be regarded as evaluation of a formal high-level design (e.g., component ADV_HLD.5 at EAL 7 with respect to the Common Criteria). Architectural evaluation is also possible at an early stage in development where the source code does not yet exist. In some cases, design information is not

shared owing to unwillingness (e.g., licencing restrictions) or unability (e.g., where the original manufacturer is out of business).

A *repository of generic attack methods* allows enumeration of all possible attacks at the level of abstraction and in the scenario of choice. Our scope is a single system, e.g., a workstation, in which benign and malicious processes co-exist. This restriction points out the contrast to large networks with many hosts – which are not our focus. Our derivation of this set of possible attacks on software architectural vulnerabilities is based on a complete enumeration of attack vectors for a Turing machine. It is cross-checked with unordered attack catalogues found in evaluation manuals, i.e., [ITS93] and [CEM04]. Introducing the theoretically derived *attack repository* allows enumeration of generalised attack methods that in the past have often existed as unordered lists or as experiential knowledge of evaluators.

Our framework consists of more than 50 Z schemas for modeling operating system structures and behaviour. In addition, we have almost as many schemas defining generic attack methods in our repository, representing approaches available to an adversary for exploiting a software's architectural vulnerabilities. (What constitutes a security breach is defined by a selection of typical security policies.) An architecural description varies in size. Our largest studied sample comprises on the order of 100 components and several hundred connectors.

Resistance classes are based on a hierarchy of security requirements and a hierarchy of attacker capabilities. Both hierarchies can be expressed by a lattice. Vulnerability is a function of risk (with *resilience* being the opposite of vulnerability). Following the widely adopted textbook definition (cf., e.g., [Gol05]), risk is a function of probability of an event occurring that leads to losses, and of the amount of possible loss. Vulnerability metrics must help to address the probability part, i.e., reducing threat (e.g., the more powerful an adversary needs to be, the less likely the threat becomes), and must help to address the losses part, i.e., reducing impact (e.g., the higher the accomplished/accomplishable security goals are, the lower the losses). Hence, *attacker capabilities* and *security requirements* chosen as a vulnerability/resilience metric is a good choice, accepted by the risk analysis and security metrics communities.

Metrics of architectural properties of software products have been extracted from established design principles and have been used in a case study. Most of the metrics are relevant in that they highlight problem areas in software development and deployment. Achieving a perfect score on a number of these metrics is a prerequisite for advancing a product to a higher resistance class. It has been demonstrated by help of the metrics that the security level of a commercial product had deteriorated from one version to the next.

We have validated usability and usefulness of our framework with data about real world attacks, including samples from [LABMC94], [Lan06b], and [LS07].

The best architectural style with respect to our metrics is a monolithic software product with a clear perimeter. Anecdotal evidence from past systems suggests that policing the perimeter is easier for a monolithic product compared to one aggregated from many distributed components.

Recalling our original research question – "Can resilience against malware attacks be assessed and rated, and if so, how?" –, we are now able to answer this question with "yes" and its six sub-questions.

What are possible axes and scales to define a security level? – We use two axes: attacker capabilities and security requirements, detailed in chapter 3.

How can a software's architecture be described, together with the surrounding operating

environment and an adversary? – We use a model of a generic computer system, detailed in chapter 4.

Which properties of an architecture can be measured and related to resilience against attacks? – We use a collection of 14 metrics that measure compliance of an architecture with established design principles for secure systems, detailed in chapter 3.

How can attacks be enumerated against which protection is required? – We use a repository of generic attacks, extracted from established evaluation criteria and we show completeness by a Turing machine approach, detailed in chapter 3.

What formal basis can be used to express and derive a security level? – We use a Z specification of our model of a generic computer system and of our attack repository, detailed in chapters 5 and 6.

Which metrics are significant and in which order should they be applied to evaluate a product? – We find that coverage of access control mechanisms, conformity of permissions, percentage of integrity-preserving operations, and coverage of trusted path mechanisms are among the more significant metrics in our collection and should be evaluated before other metrics. We validate our metrics collection with a case study, detailed in chapter 7.

8.2 Discussion

Few product security metrics have been found in previous work. With our framework for malware resilience metrics we fill a gap by ordering knowledge about software structure and attack methods and prepare it for standardisation.

Using the theoretical apparatus lead to a systematic revelation of security vulnerabilities in existing products. These could have been found without our method. However, we can claim that our analysis is comprehensive and does not leave out any architectural vulnerability of importance. Owing to the scarce information about software evaluation in its current niche market and challenged by limited repeatability and documentation of evaluation processes, it is hard to determine whether our method could speed up the software evaluation process. It takes time to create an architectural description based on an existing product; having one up front in development would reduce time spent evaluating.

Not all software vulnerabilities are expressible in the current version of the generic operating system model. Race conditions, i.e., timing problems in access control or resource reference resolution, are not adequately addressed. This probably owes to the atomic nature of Z schema operations. These ensure that parallel operations do not interfere with other processes.

Tool support for typechecking and proving revealed errors in the first versions of the specification. The type checker lead to rigour in the specification. What looked correct at first glance turned out to be subtly incorrect when compared with the ISO standard for Z. [Z00] Attempting to prove invariants of some operations, side effects of operations were revealed (and henceforth corrected). The specification became easier to use and easier to digest.

Using a freely available tool, i.e., CADi$\mathbb{Z}$, has advantages and disadvantages. Contact with the author of the tool was easy and a fix was supplied fast for a problem detected when installing und using the tool for the first time. Extensive documentation is available online. No restrictions are experienced as regards licensing (and costs) as well as using the results produced with the tool. On the downside, there is a clear lack of a large and demanding customer base. Installation is tedious, use does not conform to standard

user interface guidelines and experience. We would have preferred a tool like *ProofPower*, enhanced by the use of standard LaTeX files and better support for typesetting Z. Documentation of CADiℤ seems more oriented towards people interested in the technical details of the tool itself. It could benefit from more examples, explicit documentation of how to type Z specifications in LaTeX. Error messages of the tool are a pain in the neck and would not have been tolerated in a commercial version. However, one supposedly gets what one pays for. So, for zero direct costs the advantages by far outweigh the disadvantages.

8.3 Future work

Our work could be extended in several directions. The nature of the presented framework allows for incremental enhancements.

Security requirements could be specified at a finer level, e.g., distinguishing between protection of user data, person-related data, configuration data, and exported data along the confidentiality and integrity dimensions. However, this might lead to difficulties when comparing combined security requirement levels.

The generic operating system model could be modified to include *more connector types*, e.g., to lock access to components involved in an operation (locking as an alternative to explicit access control). *New component types* could be used as decoys to reflect such protection mechanisms as local *honey pot* components. These could make it harder for an attacker to find the correct target without being detected.

A larger number of generic attack methods in the repository, especially multi-step attacks performed over a longer time period, would yield a more comprehensive picture of the tools available to an adversary. Timing aspects, e.g., changes in access control configuration applied immediately or delayed, temporally restricted access rights, or treatment of race conditions seem like a worthwhile extension.

Measurement should be automated to gather more data more reliably in short time and at lower expense. Software architecture could be extracted from architecture design tools where available, and then transformed to fit into the structures of our model. It might also be possible to use annotated syntax trees (recovered from source code) to derive an architectural description of an application.

Graph kernel methods as outlined in [NB05] could be used to measure similarity of architectures. This could be studied in connection with changes in the level of resilience of the two architectures against malware attacks.

Availability of *better tools for the use of formal methods* for architectural analysis would be an advantage. The wish list comprises easy installation, ease of use, integration with LaTeX interoperability between tools, and documentation aimed at skilled users, not tool developers or tool experts.

Appendix A

Checklists Used to Derive Software Architecture Properties

C.C. Wood, W.W. Banks, S.B. Guarro, V.E. Hampel, and H.P. Sartorio. *Computer security: a comprehensive controls checklist.* J. Wiley & Sons, 1987 ([WBG+87])

The checklists contained in [WBG+87] emerged from a project carried out for the United States Air Force in the 1980s. They were derived from a broad examination of the then current literature and computer systems, and from discussions with computer security experts. Despite its age the questions raised are fundamental and still pertain to computer systems today.

In total the lists consist of 738 questions in 11 categories pertaining to computer security. Out of these, 5 categories were selected for further study: *Systems development*, *Data and program access*, *Input/output*, *Processing operations*, *Database and systems software*. As a result, 97 questions were evaluated as relevant as regards malware and software/system architecture. They are used in the development of software metrics for malware resistance in chapter 3. These days, computer systems do not necessarily exhibit the strong separation between development and production systems as of the time when these lists were introduced.
Relevant checklists (sections numbered as in referenced document):

- 2.2 – Systems Development
- 2.6 – Data and Program Access
- 2.7 – Input/Output
- 2.8 – Processing Operations
- 2.9 – Database and Systems Software

Relevant checklist items Items are trailed by an estimation of the lists' authors how important they are to be followed: VL Very Low, L Low, M Medium, H High, VH Very High.

2.2 – Systems Development Checklist

10. Are production programs segregated from those programs in development and in testing such that the personnel involved in these activities cannot modify, replace, or otherwise affect production programs? (VH)

14. If emergency changes to production programs are made, are such emergency changes logged, and subsequently justified and supported by appropriate documentation and management approval? (VH)

15. Are nonemergency requests for both new applications and modifications to existing programs screened by data processing management to determine their legitimacy and relative importance? (M)

16. Are all nonemergency changes to production programs initiated by and traceable to user request, management directive, or auditor recommendation? (L)

17. Are tests of vendor supplied upgrades in software performed prior to using such new software with production data? (M)

18. Are all changes to production systems accompanied by appropriate updates to existing documentation? (M)

19. Are source code compare programs or other mechanisms (such as total hash checks) used to detect unauthorized production program changes employed? (M)

20. Are all changes to production programs recorded in a protected log that reflects the content of the change, the date, and the initiating person? (H)

21. Is the ability to effect changes to production program libraries restricted to a few authorized individuals? (H)

26. Is program maintenance subject to stringent controls comparable to those found in original program development? (H)

30. Are all permanent program changes (except those provided by vendors) generated from source rather than object code? (L)

31. Are modifications to vendor software made by in-house staff kept to a minimum and completely documented? (L)

33. Are systems under development coordinated via a data dictionary (sometimes called a data directory) so that controls are consistently applied to data wherever that data may reside? (A data dictionary is a master index to data throughout an organization, not just specific application systems, computer systems, or organizational subunits.) (H)

2.6 – Data and Program Access

2. Is the identity of all computers making connection to the systems being examined, positively validated with message authentication codes, passwords, or some other similar mechanism? [Message authentication codes are cryptographically derived quantities appended to messages that are used to verify that the messages have not been modified (or in some cases deleted) in transit.] (H)

3. Are all accesses to sensitive files, databases, and programs granted only after the requesting user, process, or external system has been identified and has had its permission to access such resources verified? (VH)

4. Are all accesses to sensitive files, databases, and programs restricted by a scheme that enforces the various types of access such as read, write, delete, copy, rename, allocate, catalog, and execute (H)?

5. Is an execute-only access condition available for sensitive programs that are not to be read, copied, or otherwise used? (M)

6. Are user-created or user-controlled files, databases, and programs segregated so that no other user can access these files without first having such access specifically enabled (”superuser” or privileged user accounts excepted)? (VH)

7. Is a second identity validation control point, separate from the initial log-in identity validation, used to check file, database, and program access privileges? (The initial log-in password or other identification validation routine might be operating system based, whereas a second level of password oriented identification validation might be application supported.) (H)

10. Are sensitive nondata and nonprogram systems resources (such as terminals) controlled by passwords unique to individuals? (H)

12. Is the ability to initiate sensitive transactions controlled via an access control package or application-based control program? (H)

17. Does the actual readable (cleartext) value of passwords and other security parameters never appear in systems logs, applications logs, or other logs? (VH)

20. Are the commands to turn off or disable security-related logs strictly limited to those users with a need to invoke such commands? (H)

21. Will the turning off or disabling of security-related logs result in an entry in such logs? (H)

25. Is the file containing user passwords encrypted using a cryptographic algorithm with a high work factor (i.e., using a highly secure algorithm)? (VH)

26. If encryption is used to protect passwords in storage, is the key for the encryption process either encrypted, not on the system, or in a tamperproof security module? (To have the key in main memory in unencrypted form exposes the system to a number of very serious potential attacks.) (H)

27. Are passwords in security files encrypted with one-way functions such that cleartext (readable) password values cannot be recovered? (H)

29. Are files containing user passwords, authorization (access control) bits in records and files, authorization tables, or other mechanisms used to control user access to data and programs prevented from being modified by unauthorized users? (H)

62. Are jobs sent from one system to another controlled by user-IDs, passwords, encryption keys, message authentication codes, or other mechanisms used as access control identifiers and identity validation codes? (H)

64. If a database is used, are parts of the database restricted according to user or program privileges? (H)

69. Are access controls implemented using rules that define user privileges with regard to sensitive system resources (e.g., as a member of a given group of users, a certain user may have access to certain data by default)? (Rules that apply to several users facilitate the administration of access controls over systems resources.) (M)

72. Is the assignment of and maintenance of access constraints over files and other sensitive resources, to a large extent, handled by users rather than centralized security administrators? (In a large-scale system, centralized control of resources such as files, via security administrators, may be onerous and difficult to maintain.) (H)

82. Are user jobs prevented from reading or writing outside their assigned storage areas in main memory, on disks, and so on? (H)

83. Does the operating system erase (zeroize) all scratch space assigned to a sensitive job after the normal or abnormal termination of the job (also known as residue protection)? (H)

84. Are all mechanisms to disable access control programs (RACF™, SECURE™, TOP SECRET™, GUARDIAN™, etc.) restricted to privileged users, and does the invocation of such mechanisms result in an entry in a protected log? (VH)

85. Will a failure on the part of access control programs prevent users from accessing the system rather than no longer controlling access? (M)

87. Are computer programs that safeguard or handle sensitive information themselves safeguarded with the same or greater control than the information they apply to? (M)

91. Are microcomputers and other small computers protected with logical access control packages somewhat like RACF™ and ACF2™? (Often smaller machines have no such software available and/or the users mistakenly believe that it is not necessary for these machines.) (H)

92. Within the organization, is one standard access control package used on all large-scale machines running the same or similar operating systems? (Secure transfer of sensitive files and other controlled systems resources will be significantly facilitated if this is the case.) (H)

93. Is access to particular sensitive files or databases restricted by allowing only certain authorized programs to access such files or databases? (Some call this "restricted paths" to the data.) (M)

98. Did the installation of the access control package in use, such as RACF™ and ACF2™, not require that the operating system be modified? (Operating system modification may unwittingly open up additional vulnerabilities.) (L)

99. Does the access control package in use have a control philosophy that all resources (tapes, files, etc.) are restricted by default rather than unrestricted by default? (If restrictions are forgotten for a sensitive resource, this philosophy will mean that the resource is protected, whereas with the countervailing philosophy, the resource will not be protected.) (VH)

101. Does the access control package in use provide access protection (read, write, execute, delete, etc.) for separate files resident on tape? (Some access control packages protect only the tape volume, not the specific files contained on the tape volume.) (M)

103. Does the access control package in use have the ability to protect catalogs (directories of locations of files and libraries) rather than just files? (Program change control processes and certain other controls may be supported by such a facility.) (H)

105. Can changes in the access control system's rules and passwords be made without re-IPLing (again initial program loading) the system? (Some systems require that the system come down and then be regenerated to change these critical parameters, although it is preferable to be able to do it as operations continue.) (M)

2.7 – Input/Output

1. Do application programs include input edit routines such as character type checks, limit checks, validity checks, sequence check, reasonableness checks, consistency checks, or other similar types of checks? (VH)

2. Is redundant entry of data by another operator (e.g., verification of keypunching or keytaping) performed as a means to reduce errors in input data? (M)

3. Are the operators inputting data required to enter critical fields twice as a means of ensuring the correctness of such data? (M)

15. Are critical or very sensitive input transactions removed from the normal flow of input and handled with special procedures, such as separate input item serial numbers? (H)

25. Is the work of all persons involved in data input and output traceable to such individuals? (VH)

26. Are the types of transactions that users can input restricted to those directly related to their job duties? (H)

2.8 – Processing Operations

6. Are protected copies of recently outdated production programs kept conveniently available such that they may be used in the event that newer versions do not run properly? (L)

8. Are production programs statically linked together so that dynamic linkages at run time do not represent an opportunity for unauthorized replacement of some of these programs? (The interfaces and calls between production programs should not be subject to modification at program invocation time, to do so is to open up the possibility that one of these production programs could be replaced with an unauthorized program.) (M)

10. Are control totals that balance the entire system, or non-application resources therein, used (e.g., balancing between systems and balancing/reconciling files)? (These control totals are particularly important when several interdependent applications are run sequentially.) (VH)

15. Are procedures for blanking or purging intermediate storage (including scratch tapes and disks) always carried out following jobs using sensitive information? (VH)

16. Is the reading of scratch tapes and other intermediate storage prohibited unless the same job/user has previously written to the same tape or intermediate storage media? (M)

29. Are special routines used to periodically recalculate hash totals of production programs to ensure that unauthorized modifications have not been made? (A hash total in this instance serves as a file authentication code in that it is a number resulting from a calculation that includes the entire production program as input.) (M)

30. To detect unauthorized modifications, is a program that compares controlled duplicates of production programs with the current on-the-system production programs used? (M)

31. If the system handles very sensitive data, is it run only during those hours when management is on the premises to supervise? (M)

33. Is a system in effect that controls changes to production programs and that logs all changes to production applications software, operations software, and operating systems software? (VH)

35. Is invocation of the programs in privileged libraries always followed by an entry in a secure log? (VH)

36. Are access restrictions in effect on systems utilities (such as Superzap or Disk Analyzer) that might be used to manipulate the system or otherwise circumvent controls? (VH)

46. Are all programs required to be run under the control of an operating system that automatically and consistently records their use on the console log? (M)

54. Is a secure log of all changes to security tables (such as the tables that permit users to access certain system resources) kept? (VH)

56. Are master commands, such as boot the system, restricted to a small number of terminals and a small number of operators? (H)

57. To protect sensitive information held in main memory from inadvertent disclosure (as might take place when a memory dump occurs), are these data cryptographically protected at least part of the time? (H)

59. Are the job assignments of computer operations personnel rotated periodically? (H)

2.9 – Database and Systems Software

5. Is access to both databases and data dictionaries restricted to users and processes that have a need to know? (VH)

9. Are authorization tables, profiles, or some similar method used to restrict the access of users and processes to only those transactions for which they have been specifically authorized? (VH)

13. Are all inquiries and updates to databases recorded in secure logs (i.e., are audit trails provided)? (VH)

14. Does the database's logging system record all deleted data, all new data, the origin of all transactions, application utilization of certain databases, user utilization of certain databases, security violations, and various related accounting information? (H)

17. If an attempt to perform unauthorized actions is in progress, will a DBA [database administrator] or some other staff member in a position to take action be notified in real time? (M)

21. Does database software clear buffers after transaction processing to prevent unauthorized access to the contents of such buffers? (L)

22. Are sensitive database fields encrypted to prevent unauthorized access? (H)

25. When special DBMS utilities or DBA-privileged commands are being invoked, will these instructions be followed only if entered from a specific master terminal? (L)

34. When databases are not under the control of DBMS routines, are there mechanisms to prevent access to such databases (particularly to protect against unauthorized copying)? (M)

45. Are compilers, interpreters, and other language translators that may facilitate program development not resident on computers that are used for production jobs only? (M)

46. Are text editors and other utilities that might be used to make unauthorized modifications to software not resident on computers that are used for production jobs only? (M)

50. Is access to sensitive systems utilities strictly controlled such that only systems programmers and others with a need are granted such access? (VH)

53. To make unauthorized modification extremely difficult, are particularly sensitive programs implemented in firmware as opposed to software? (Firmware cannot be modified except with sophisticated equipment, and even then may have to be removed from the device that houses it.) (M)

54. To detect unauthorized modification of systems software, are comparator programs used to compare system resident copies of sensitive software with copies securely maintained elsewhere (may be used for both object and source copies)? (L)

55. To detect unauthorized modification of important software, are check-sum counts of the bits contained in software packages compared, or are hash totals of such software packages compared? (L)

56. Are check-sums, hash totals, cryptographic authentications, or some other technique used to ensure that sensitive internal tables are not tampered with?

58. Does the operating system use kernelized software (restricted access software for which special security validation work has been done)? (VH)

59. Does the systems software support multilevel security, that is, are users with different security clearances and data with different sensitivity classifications concurrently handled in such a way that proper separation is maintained? (VH)

61. To initiate a new processing day, and to assure that no unauthorized changes to sensitive software have been made, is such software down-line loaded from a central to remote sites? (VL)

65. Does systems software or hardware prevent one user from gaining access to another user's storage space in buffers, main memory, disks, and so on? (H)

68. Is encryption used to safeguard particularly sensitive systems software and data? (H)

69. If a computer system is to be subsequently used by another party with lower-level clearance or privileges, must internal memory be erased, all sensitive storage media removed, and all proprietary systems and otherwise sensitive software removed? (M)

70. If a communications circuit fails, or if some other event severs a connection with an on-line process/user, will system software require full identification and authentication from the next process/user coming in on that line/port? (M)

72. Is the ability to invoke mechanisms, used to bypass security software(e.g., use of disk error analysis routines to bypass file access control software) strictly limited to authorized persons? (VH)

73. Will an abend (abnormal ending) to a program that utilizes sensitive information protected by access control software be prevented from resulting in a dump of such sensitive information? (M)

76. Does policy prohibit system programmers from placing "hooks" (e.g., places where an unauthorized program might be called or where an unauthorized transaction might be initiated) in their programs that may later be used to compromise production program security? (M)

77. Is the operating system log (also called an audit trail) protected such that unauthorized modification is exceedingly difficult? (VH)

78. Are commands that turn off operating system logs very strictly controlled, and does invocation of such a command also result in an entry in these logs? (VH)

79. Is the use of abort and cancel instructions limited so that users cannot thereby enter executive (privileged) mode? (On certain systems these commands may be used to exit application programs, and thereby to circumvent the control provided by such application systems.) (H)

81. Is a program library management system used to control modifications to and usage of systems software? (M)

CORAS methodology. [modified 2004-10-17, down-loaded 2005-05-23]. http://coras.sourceforge.net/, 2004 ([COR04])

The questionnaires contained in [COR04] were used in the EU research project *CORAS* (*A Platform for Risk Analysis of Security Critical Systems*). Here, model-based risk assessment was supported by a UML-based specification language, a library with collected experiences, a software tool, and an XML-based specification for data exchange about security-critical systems. With the list, an operator of the tools is aided in addressing the security aspects of the system under evaluation.

In total the lists consist of 176 questions in 8 categories pertaining to computer security on various levels (high level management down to technical aspects). Out of these, 5 categories were selected for further study: *Human*, *Physical*, *Information*, *Software*, *Exceptional circumstances*. As a result, 51 questions were evaluated as relevant as regards malware and software/system architecture. They are used in the development of software metrics for malware resistance in chapter 3.

Relevant questionnaires (sections numbered as in referenced document):

- 2 – Human
- 3 – Physical
- 4 – Information
- 7 – Software
- 8 – Exceptional circumstances

Relevant questionnaire questions

2 – Vulnerability-questionnaire: Human

8. How will a user recognize a security breach or security anomaly?
12. Are there any procedures or routines for testing that security handling is functioning and effective?
13. Are there routines for reporting software malfunctions?

3 Vulnerability-questionnaire: Physical

17. Is access to computer resources restricted at the operating system level?
18. Is terminal identification used to authenticate connections to specific locations?
19. Are there secure log-on procedures in order to avoid unauthorised users to access information services?
20. Are users identified and authenticated?
22. Is the use of system utilities restricted to authorized personnel?
23. Are users automatically logged off after a certain time of inactivity?
24. Are there limitations on connection times for high-risk applications?

4 Vulnerability-questionnaire: Information

19. Are backups of information taken on a regular and scheduled basis and stored with an appropriate level of physical and environmental protection?
21. Are system faults logged?
22. Is encryption used in order to protect highly sensitive data?
23. Is message authentication (by use of digital signatures or otherwise) used for applications, which involve the transmission of sensitive data?
25. Are cryptographic keys properly protected and managed?
30. Are there mechanisms that prevent users from installing software that is downloaded from unprotected or untrustworthy sources?
41. Are systems monitored in order to detect unauthorised activities?
43. Are input data validated?
44. Are data processed by application systems validated?
45. Is message authentication used to detect unauthorised changes to or corruption of the contents of a transmitted electronic message?
46. Is data output from an application system validated?
47. Are there control routines for protection and verification of system files?

48. Are there control routines for protection and verification of software on operational systems?

49. Are there formal procedures for change control in development and support processes?

51. Are there procedures to document modifications of software packages or deployment of new packages?

52. Are there procedures in place to ensure that a deployed system is used in the same manner as the designers thought it would be used?

7 Vulnerability-questionnaire: Software

8. Are there formal procedures controlling the access to information systems and services?

10. Are allocation and use of privileges restricted and controlled?

13. Are all connections to network services controlled?

14. Is a users network access limited as to give access only to services that are strictly needed?

15. Is the route from a users terminal to the computer service controlled?

16. Are users that are using external connections adequately authenticated?

17. Are external terminals or systems connected to the host system adequately authenticated?

18. Are diagnostic/remote maintenance ports and systems securely protected?

19. Is the network divided into sub networks in order to provide added protection?

20. Are business application access control policies reflected in limitations on network connections?

21. Are unnecessary network services disabled or monitored?

22. Is access to software and application appropriately controlled?

24. Are input data to application systems validated?

25. Are data processed by application systems validated?

26. Is message authentication used to detect unauthorised changes to or corruption of the contents of a transmitted electronic messages?

27. Is data output from an application system validated?

28. Are there control routines for protection and verification of system files?

29. Are there control routines for protection and verification of software on operational systems?

30. Are there formal procedures for change control in development and support processes?

31. Are the pre-conditions for the provided interfaces and post-conditions of the required interfaces documented?

32. Does the component verify that the pre-conditions are met before executing the functionality? Has any pre-condition verification been tested?

33. Were invariants identified and documented? Were invariants verified during testing with assertions?

34. Are there Exception Handling provisions for software components? Are unhandled exceptions documented for each component that throws them? Are the available exception handlers tested?

8 Vulnerability-questionnaire: Exceptional circumstances

9. To what extent can the TOE function autonomously (in whole or in part) with respect to the environment (e.g. without a WAN, if law and order breaks down, during massive and prolonged power outages, if all key personnel are disabled etc.)?

11. To what extent is the TOE designed to be fail-soft in exceptional circumstances?

12. What are the trigger criteria that will cause the TOE to fail-safe, i.e. enter a stable/safe/static/known state? How will the fail-safe state be achieved?

Appendix B

StarMoney architecture

StarMoney 5.0 consists of more than 2,000 individual files comprising stand-alone executables, dynamically linked libraries, user interface definition files, configuration files, database files. In addition, some configuration data is stored in the system registry.

Architecturally, the homebanking application *StarMoney 5.0* with a total size of ca. `50 MB` consists of 3 small executable modules, `StartStarMoney.exe [60 KB]` for preparing the start of the main executable `StarMoney.exe [172 KB]`, and `SCRSetup.exe` for configuring smart card readers attached to the system. (5 additional executable modules perform peripheral functionality: Conversion of older versions' data (`smkonv.exe`), T-Online Classic access (`CLGate32.exe`, `sfkclgateslave.exe`, `sfktonac.exe`), remote support (`NetViewer.exe`))

Dependencies between modules are extracted by syntactic analysis of the binaries by own tools and semantic analysis of fixed run-time bindings, e.g., using the `Dependency Walker` tool for Microsoft Windows executables (included, e.g., with *Visual Studio*). Fixed bindings are decided at compile time by partial reference, i.e. filename only, and resolved at run time. Resolution out of control of the application may lead to executables being loaded that were not intended. Dynamic bindings are under control of the application, i.e., it can decide whether to operate with complete path information to a file or to defer resolution to the operating system. Here, we can use, e.g., `FileMon` and `RegMon` tools to obtain information about the bindings. These tools are provided by `sysinternals.com`.

The executable modules are supported by 89 executable library files that extend the functionality of the main executable. They can be partitioned into 6 function groups: *Core banking* (18 libraries), *Smart card communication* (4 libraries), *Database access* (11 libraries), *Communication* (17 libraries), *User interface* (24 libraries), *Miscellaneous* (15 libraries). We list size, name, and purpose of each identified module.

Core banking 18 files

Size	Name	Purpose
94,208	sfkcepthdl.dll	T-Online CEPT communication
221,184	sfkhbcihdl.dll	HBCI routines
110,592	sfkhbcipthdl.dll	HBCI protocol handler
90,112	sfkhbcitranslator.dll	HBCI protocol routines
36,864	sfkinternalreqhdl.dll	Securities trading
81,920	sfkkernel.dll	HBCI kernel
28,672	sfkmp940.dll	SWIFT MT 940 parser
28,672	sfkmpapo.dll	MT 940 parser APO bank
28,672	sfkmpizb.dll	MT 940 parser IZB computing center

28,672	sfkmpkordob.dll	MT 940 parser Kordoba Payment Systems
28,672	sfkmpvbv.dll	MT 940 parser VBV
65,536	sfkmtbase.dll	SWIFT message types parser
45,056	sfkparameterdatahdl.dll	Bank parameter data handler
200,704	sfkparser.dll	Transaction content parser
73,728	sfksmgatewayhdl.dll	Banking gateway
503,808	sfmcorecoapp.dll	Application coordination
688,128	sfmcorecostock.dll	Stock trading
139,264	sfmcorecotimedep.dll	Time deposits

Smart card communication and smart card reader 4 files

20,480	pcsckrnlini.dll	PC/SC configuration for HBCI
139,264	sfkctapicard.dll	CT-API compatible card reader
135,168	sfkpcscard.dll	PC/SC compatible card reader
196,608	sfksecmod.dll	Security module creation

Database access 11 files

503,808	sfbsdb.dll	Application database access
53,248	sfmcorecosystemdb.dll	System database access
327,680	sfmcoredb.dll	Application database access
221,184	sfmcoreimex.dll	Data import/export
49,152	sfmcoreimporterimpl.dll	Import coordination
90,112	sfmcoreimportersc2.dll	Data import S-Connect
135,168	sfmcoreimporterzvlight.dll	Data import ZV light
204,800	sfmpreconvzvl.dll	Data conversion ZV light
331,776	sfmuidetailadmin.dll	Application database administration
163,840	sfmuidetailbadmin.dll	Application database administration
135,168	sfmuidetailimex.dll	Data import/export

Communication 17 files

336,384	aol32.dll	AOL connection — 3rd party
49,152	sfbcore.dll	Character encoding/conversion
118,784	sfbcrypto.dll	Cryptographic operations
286,720	sfbinet.dll	Internet data transfer
53,248	sfbosl.dll	IPC and network communication
225,280	sfbtoolsn.dll	File and network access
94,208	sfbvos.dll	Socket communication
24,576	sfkclgateaccess.dll	T-Online gateway connection
65,536	sfkprotocolhdl.dll	HTTP protocol handler
28,672	sfktonac.dll	T-Online controller
73,728	sfktonlinedec.dll	T-Online macro processor
147,456	sfkwebbankinghdl.dll	Screen scraping engine
147,456	sfkwebengine.dll	Web engine
172,032	sfkwebenginecmd.dll	Web engine commands

32,768	`sfkxmlhttprequest.dll`	HTTP request factory
507,904	`CATClient\cptdll32.dll`	T-Online CEPT
90,112	`CATClient\TrpS2032.dll`	T-Online Transport/S

User interface 24 files

57,344	`iecontr.dll`	IE Hosting
188,416	`libtidy.dll`	Tidy HTML routines — 3rd party
461,864	`olch2d32.dll`	Chart control — 3rd party [signed]
856,064	`sfbsv.dll`	Starview control handling
1,183,744	`sfbsvtools.dll`	JPEG image handling
2,117,632	`sfmcorecobasic.dll`	User interface coordination
90,112	`sfmcorere.dll`	Reporting routines
950,272	`sfmuicontrol.dll`	Web browser control host
1,740,800	`sfmuidetailaccount.dll`	Account details and dispatch
1,028,096	`sfmuidetailbasic.dll`	High-level dispatcher
430,080	`sfmuidetailcommon.dll`	Shared dialogs
131,072	`sfmuidetailhbauto.dll`	Transaction automation
397,312	`sfmuidetailoverview.dll`	Account details
397,312	`sfmuidetailreport.dll`	Report configuration
675,840	`sfmuidetailstock.dll`	Securities trading
180,224	`sfmuidetailtimedep.dll`	Time deposits
978,944	`sfmuidetailtransfer.dll`	Money transfer
53,248	`sfmuief.dll`	Browser properties wrapper
32,768	`sfmuihtmlhelp.dll`	HTML Help loader
40,960	`sfmuiie.dll`	Browser methods wrapper
45,056	`sfmuiimporterimpl.dll`	Data import
49,152	`sfmuiimportersc2.dll`	Data import S-Connect
36,864	`sfmuiimportershare.dll`	Data import shared dialogs
36,864	`sfmuiimporterzvlight.dll`	Data import ZV light

Miscellaneous 15 files

401,462	`msvcp60.dll`	MS C++ Runtime Library — 3rd party
202,240	`patchw32.dll`	Software updates — 3rd party
36,864	`sfbmsutil.dll`	Stream conversions
28,672	`sfbsolar.dll`	Intra-memory data handling
151,552	`sfbtoolso.dll`	Languages and format conversion routines
57,344	`sfbunzip.dll`	Data compression
24,576	`sfbversioninfo.dll`	File version information
45,056	`sfbxmlutil.dll`	XML parser
335,872	`sfktools.dll`	Routines collection
77,824	`sfkupdatehdl.dll`	Software updates
57,344	`sfkxinclude.dll`	XML routines
32,768	`sfkxsltscripthdl.dll`	XSLT script handler
139,264	`sfmcoremisc.dll`	Diverse routines
237,568	`sfmuionlactiv.dll`	Online activation

`81,920 sfmuisvbrbx.dll` Browse box

Where we omit file extensions in our list, the extension is `.dll`. The first column gives the module loading another. A host module in parentheses is one loaded by the initial host module. The second column gives the loaded module and its location. Usual abbreviations for folders are used, e.g. `%WINSYSDIR%` for the main operating system folder for libraries. Subsequent files in the same folder are shown indented. The third column states if loading happens with a complete or incomplete reference (omitted in cases where the operating system is responsible for sequences of loaded system libraries). The fourth and final column shows the entity responsible for resolving an incomplete reference, and can be the operating system – *OS* – or the *application*.

The discovered module dependencies of *StarMoney* are:

Main executables 8 files

Host module	Hosted/imported module	Reference	Resolution
StartStarMoney.exe	%WINSYSDIR%\kernel32	incomplete	OS
(kernel32)	%WINSYSDIR%\ntdll	(OS)	
	%WINSYSDIR%\user32	incomplete	OS
(user32)	%WINSYSDIR%\gdi32	(OS)	
(user32)	%WINSYSDIR%\kernel32	(OS)	
(user32)	%WINSYSDIR%\ntdll	(OS)	
	%WINSYSDIR%\advapi32	incomplete	OS
(advapi32)	%WINSYSDIR%\ntdll	(OS)	
(advapi32)	%WINSYSDIR%\kernel32	(OS)	
(advapi32)	%WINSYSDIR%\rpcrt4	(OS)	
(rpcrt4)	%WINSYSDIR%\ntdll	(OS)	
(rpcrt4)	%WINSYSDIR%\kernel32	(OS)	
(rpcrt4)	%WINSYSDIR%\advapi32	(OS)	
	%WINSYSDIR%\ole32	incomplete	OS
(ole32)	%WINSYSDIR%\advapi32	(OS)	
(ole32)	%WINSYSDIR%\gdi32	(OS)	
(ole32)	%WINSYSDIR%\kernel32	(OS)	
(ole32)	%WINSYSDIR%\msvcrt	(OS)	
(ole32)	%WINSYSDIR%\ntdll	(OS)	
(ole32)	%WINSYSDIR%\rpcrt4	(OS)	
(ole32)	%WINSYSDIR%\user32	(OS)	
	%WINSYSDIR%\gdi32	incomplete	OS
(gdi32)	%WINSYSDIR%\kernel32	(OS)	
(gdi32)	%WINSYSDIR%\ntdll	(OS)	
(gdi32)	%WINSYSDIR%\user32	(OS)	
	%WINSYSDIR%\msvcrt	incomplete	OS
(msvcrt)	%WINSYSDIR%\kernel32	(OS)	
(msvcrt)	%WINSYSDIR%\ntdll	(OS)	
	%INSTDIR%\msvcp60	incomplete	OS
(msvcp60)	%WINSYSDIR%\msvcrt	incomplete	OS
(msvcp60)	%WINSYSDIR%\kernel32	incomplete	OS
	%INSTDIR%\sfbunzip	incomplete	OS

(sfbunzip)	%WINSYSDIR%\msvcrt	incomplete	OS
(sfbunzip)	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\wsock32	incomplete	OS
	%INSTDIR%\StarMoney.exe	incorrect	Application
	smoney.ini	complete	Application
	ouext.ini	complete	Application
	bootload.lst	complete	Application
	banner.ini	unknown	
	pocket.ini	unknown	
	setup.exe	unknown	

We found that operating system libraries that are loaded upon module execution, are always referenced by their file name only, omitting path information. In the following, we will not list these recursively for brevity reasons. Hence, dependencies on operating system modules are only listed for the modules originally loaded by *StarMoney*. Libraries in turn loaded with incomplete reference by operating system libraries are not shown. These have to be determined by analysing the internal dependencies of operating system files, especially those stored in the `%WINSYSDIR%` folder.

Host module	Hosted/imported module	Reference	Resolution
StarMoney.exe	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\shell32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	%INSTDIR%\sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfkkernel	incomplete	OS
	sfmcorecoapp	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmuicontrol	incomplete	OS
	sfmuidetailbasic	incomplete	OS
	sfmuief	incomplete	OS
	sfmuiie	incomplete	OS
	sfmcoremisc	incomplete	OS
	%INSTDIR%_sm.exe	unknown	
	%INSTDIR%\netviewer.exe	unknown	
	sfmuibpdview	incomplete	OS
	sfmuidbview	incomplete	OS
	%IEDIR%\iexplore.exe	unknown	
	%INSTDIR%\euro.rat	complete	Application
	%INSTDIR%\smoney.ini	complete	Application
	.\agent\smosetup.ini	complete	Application
	%INSTDIR%\sfmuilang49	unknown	

Host module	Hosted/imported module	Reference	Resolution
	instances.ini	unknown	
	ouext.ini	unknown	
	pocket.ini	unknown	

Host module	Hosted/imported module	Reference	Resolution
SCRSetup.exe	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%INSTDIR%\sfbtoolsn	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbsv	incomplete	OS
	sfbsvtools	incomplete	OS
	sfktools	incomplete	OS
	pcsckrnlini	incomplete	OS
	sfkpcscard	incomplete	OS
	sfkctapicard	incomplete	OS
	%WINDIR%\hbcikrnl.ini	complete	Application

Host module	Hosted/imported module	Reference	Resolution
smkonv.exe	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%INSTDIR%\sfbtoolsn	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfmuiimporterimpl	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
CLGate32.exe	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\gdi32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\shell32	incomplete	OS
	%WINSYSDIR%\comctl32	incomplete	OS
	%WINSYSDIR%\comdlg32	incomplete	OS
	%INSTDIR%\CATClient\CptDll32	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfkclgateslave.exe	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%INSTDIR%\CATClient\CLGate32.exe	unknown	

Host module	Hosted/imported module	Reference	Resolution
sfktonac.exe	%INSTDIR%\CATClient\sfktonac	incomplete	OS
	%WINSYSDIR%\krnl386.exe	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
NetViewer.exe	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\oleaut32	incomplete	OS
	%WINSYSDIR%\version	incomplete	OS
	%WINSYSDIR%\gdi32	incomplete	OS
	%WINSYSDIR%\ole32	incomplete	OS
	%WINSYSDIR%\comctl32	incomplete	OS
	%WINSYSDIR%\shell32	incomplete	OS
	%WINSYSDIR%\winmm	incomplete	OS
	%WINSYSDIR%\wsock32	incomplete	OS

Core banking 18 files

Host module	Hosted/imported module	Reference	Resolution
sfkcepthdl.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfktools	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfkhbcitranslator	incomplete	OS
	sfbvos	incomplete	OS
	sfkhbcihdl	incomplete	OS
	sfkprotocolhdl	incomplete	OS
	sfktonlinedec	incomplete	OS
	sfkcepthdl	incomplete	OS
	sfkmeide	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfkhbcihdl.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfkhbcitranslator	incomplete	OS
	sfktools	incomplete	OS
	sfkprotocolhdl	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfbvos	incomplete	OS
	sfkctapicard	incomplete	OS
	sfksecmodrm	incomplete	OS
	sfksecmod	incomplete	OS
	sfkpcscard	incomplete	OS

	connect.ini	unknown	
	versions.ini	unknown	
	creader.ini	unknown	

Host module	Hosted/imported module	Reference	Resolution
sfkhbcipthdl.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfkhbcitranslator	incomplete	OS
	sfktools	incomplete	OS
	sfkprotocolhdl	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfkhbcihdl	incomplete	OS
	%INSTDIR%\smoney.ini	complete	Application

Host module	Hosted/imported module	Reference	Resolution
sfkhbcitranslator.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfktools	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfbcore	incomplete	OS
	sfbcrypto	incomplete	OS
	sfbvos	incomplete	OS
	sfkmeide	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfkinternalreqhdl.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfktools	incomplete	OS
	sfkprotocolhdl	incomplete	OS
	sfbtoolsn	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfkkernel.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfktools	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfbvos	incomplete	OS
	sfbinet	incomplete	OS
	sfbsolar	incomplete	OS
	sfkparameterdatahdl	incomplete	OS
	sfkparser	incomplete	OS
	sfktonlinedec	incomplete	OS
	sfkwebbankinghdl	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
	sfkinternalreqhdl	incomplete	OS
	sfkcepthdl	incomplete	OS
	sfksmgatewayhdl	incomplete	OS
	sfkhbcipthdl	incomplete	OS
	sfkhbcihdl	incomplete	OS
	sfkupdatehdl	incomplete	OS
	%WINSYSDIR%\rasapi32	incomplete	OS
	aol.exe	unknown	
	aolhb32.exe	unknown	
	%INSTDIR%\aol32	incomplete	OS
	connect.ini	unknown	

Host module	Hosted/imported module	Reference	Resolution
sfkmp940.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfktools	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfkmtbase	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfkmpapo.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfktools	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfkmtbase	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfkmpizb.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfktools	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfkmtbase	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfkmpkordob.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfktools	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfkmtbase	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfkmpvbv.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfktools	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfkmtbase	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfkmtbase.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfktools	incomplete	OS
	sfbtoolsn	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfkparameterdatahdl.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\oleaut32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbcore	incomplete	OS
	sfkhbcitranslator	incomplete	OS
	sfktools	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfbvos	incomplete	OS
	sfbmsutil	incomplete	OS
	bpdhandler.ini	complete	Application

Host module	Hosted/imported module	Reference	Resolution
sfkparser.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfktools	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfbvos	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfksmgatewayhdl.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfkhbcitranslator	incomplete	OS
	sfktools	incomplete	OS
	sfkprotocolhdl	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfkhbcihdl	incomplete	OS
	sfbsolar	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmcorecoapp.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfkkernel	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
	sfbcrypto	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcorecosystemdb	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmcoremisc	incomplete	OS
	sfmcorecostock	incomplete	OS
	sfmcorecotimedep	incomplete	OS
	%WINSYSDIR%\wsock32	incomplete	OS
	%INSTDIR%\smoney.ini	complete	Application
	%INSTDIR%\agent\onlineup.ini	unknown	
	\Banner\banner.ini	unknown	
	\Banner\rpdid.ini	unknown	

Host module	Hosted/imported module	Reference	Resolution
sfmcorecostock.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfkkernel	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmcoremisc	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmcorecotimedep.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmcoremisc	incomplete	OS

Smart card communication and smart card reader 4 files

Host module	Hosted/imported module	Reference	Resolution
pcsckrnlini.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%WINSYSDIR%\scarddlg	incomplete	OS
	%WINSYSDIR%\winscard	incomplete	OS
	%WINDIR%\hbcikrnl.ini	complete	Application

Host module	Hosted/imported module	Reference	Resolution
sfkctapicard.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfktools	incomplete	OS
	sfbvos	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfbcrypto	incomplete	OS
	ct32	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfkpcscard.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbcrypto	incomplete	OS
	%WINSYSDIR%\winscard	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfksecmod.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbcrypto	incomplete	OS
	sfktools	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfbvos	incomplete	OS
	sfkparameterdatahdl	incomplete	OS
	sfkpcscard	incomplete	OS
	sfkctapicard	incomplete	OS
	sfkremotecard	incomplete	OS
	creader.ini	unknown	
	connect.ini	unknown	

Database access 11 files

Host module	Hosted/imported module	Reference	Resolution
sfbsdb.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\ole32	incomplete	OS
	%WINSYSDIR%\oleaut32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%WINSYSDIR%\odbc32	incomplete	OS
	%WINSYSDIR%\odbccp32	incomplete	OS
	%INSTDIR%\sfbtoolso	incomplete	OS

	sfbtoolsn	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	%INSTDIR%\smoney.ini	complete	Application

Host module	Hosted/imported module	Reference	Resolution
sfmcorecosystemdb.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmcoremisc	incomplete	OS
	sfmcorecobasic	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmcoredb.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbsdb	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfmcoremisc	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmcoreimex.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsv	incomplete	OS
	sfbsdb	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmcoremisc	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmcoreimporterimpl.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS

```
                           sfktools                 incomplete  OS
                           sfbsv                    incomplete  OS
                           sfbsdb                   incomplete  OS
                           sfmcorecobasic           incomplete  OS
                           sfmcoredb                incomplete  OS
                           sfmcoremisc              incomplete  OS
                        *.ini                       complete    User

Host module             Hosted/imported module     Reference   Resolution
sfmcoreimportersc2.dll  %WINSYSDIR%\kernel32       incomplete  OS
                        %WINSYSDIR%\msvcrt         incomplete  OS
                        %INSTDIR%\sfbtoolso        incomplete  OS
                           sfbtoolsn                incomplete  OS
                           sfktools                 incomplete  OS
                           sfbsvtools               incomplete  OS
                           sfbsv                    incomplete  OS
                           sfmcorecobasic           incomplete  OS
                           sfmcoremisc              incomplete  OS
                           sfmcorecotimedep         incomplete  OS
                           sfmcorecostock           incomplete  OS
                           sfmcorecoapp             incomplete  OS
                           sfmcoredb                incomplete  OS
                           sfmcoreimporterimpl      incomplete  OS

Host module             Hosted/imported module     Reference   Resolution
sfmcoreimporterzvlight.dll  %WINSYSDIR%\kernel32  incomplete  OS
                        %WINSYSDIR%\oleaut32       incomplete  OS
                        %WINSYSDIR%\msvcrt         incomplete  OS
                        %INSTDIR%\msvcp60          incomplete  OS
                           sfbtoolso                incomplete  OS
                           sfbtoolsn                incomplete  OS
                           sfktools                 incomplete  OS
                           sfbsvtools               incomplete  OS
                           sfbsv                    incomplete  OS
                           sfmcorecobasic           incomplete  OS
                           sfmcoremisc              incomplete  OS
                           sfmcorecoapp             incomplete  OS
                           sfmcoredb                incomplete  OS
                           sfmcoreimporterimpl      incomplete  OS
                           sfmpreconvzvl            incomplete  OS

Host module             Hosted/imported module     Reference   Resolution
sfmpreconvzvl.dll       %WINSYSDIR%\kernel32       incomplete  OS
                        %WINSYSDIR%\advapi32       incomplete  OS
                        %WINSYSDIR%\user32         incomplete  OS
                        %WINSYSDIR%\ole32          incomplete  OS
                        %WINSYSDIR%\oleaut32       incomplete  OS
                        %WINSYSDIR%\gdi32          incomplete  OS
```

	%WINSYSDIR%\shell32	incomplete	OS
	%WINSYSDIR%\comctl32	incomplete	OS
	%WINSYSDIR%\comdlg32	incomplete	OS
	*.ini	complete	User

Host module	Hosted/imported module	Reference	Resolution
sfmuidetailadmin.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfkkernel	incomplete	OS
	sfmcorecoapp	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmcoremisc	incomplete	OS
	sfmuicontrol	incomplete	OS
	sfmuisvbrbx	incomplete	OS
	sfmuidetailbasic	incomplete	OS
	oflagent.exe	unknown	

Host module	Hosted/imported module	Reference	Resolution
sfmuidetailbadmin.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfmcorecoapp	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcorecosystemdb	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmcoremisc	incomplete	OS
	sfmuicontrol	incomplete	OS
	sfmuidetailbasic	incomplete	OS
	sfmuisvbrbx	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmuidetailimex.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS

	sfbsv	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcorecostock	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmcoreimex	incomplete	OS
	sfmuicontrol	incomplete	OS
	sfmuidetailbasic	incomplete	OS
	sfmuiie	incomplete	OS

Communication 17 files

Host module	Hosted/imported module	Reference	Resolution
aol32.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\ole32	incomplete	OS
	%WINSYSDIR%\oleaut32	incomplete	OS
	%WINSYSDIR%\gdi32	incomplete	OS
	%WINSYSDIR%\shell32	incomplete	OS
	%WINSYSDIR%\comctl32	incomplete	OS
	%WINSYSDIR%\lz32	incomplete	OS
	aolhb32	unknown	
	%WINSYSDIR%\imm32	incomplete	OS
	vcltest3	unknown	
	%WINSYSDIR%\psapi	unknown	
	%WINSYSDIR%\vdmdbg	incomplete	OS
	%WINSYSDIR%\ntvdm.exe	unknown	
	waol.exe	unknown	
	aol.exe	complete	Application
	%WINDIR%\win.ini	complete	Application

Host module	Hosted/imported module	Reference	Resolution
sfbcore.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbvos	incomplete	OS
	sfbosl	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfbcrypto.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfbinet.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbcrypto	incomplete	OS

```
                        sfbosl                 incomplete  OS
                        sfbsolar               incomplete  OS
                        sfbtoolsn              incomplete  OS
                        sfbvos                 incomplete  OS

Host module             Hosted/imported module   Reference   Resolution
sfbosl.dll              %WINSYSDIR%\kernel32     incomplete  OS
                        %WINSYSDIR%\user32       incomplete  OS
                        %WINSYSDIR%\advapi32     incomplete  OS
                        %WINSYSDIR%\msvcrt       incomplete  OS
                        %WINSYSDIR%\wsock32      incomplete  OS
                        %WINSYSDIR%\mpr          incomplete  OS

Host module             Hosted/imported module   Reference   Resolution
sfbtoolsn.dll           %WINSYSDIR%\kernel32     incomplete  OS
                        %WINSYSDIR%\user32       incomplete  OS
                        %WINSYSDIR%\shell32      incomplete  OS
                        %WINSYSDIR%\mpr          incomplete  OS
                        %WINSYSDIR%\msvcrt       incomplete  OS
                        %INSTDIR%\msvcp60        incomplete  OS
                           sfbvos                incomplete  OS

Host module             Hosted/imported module   Reference   Resolution
sfbvos.dll              %WINSYSDIR%\kernel32     incomplete  OS
                        %WINSYSDIR%\msvcrt       incomplete  OS
                        %INSTDIR%\msvcp60        incomplete  OS
                           sfbosl                incomplete  OS

Host module             Hosted/imported module   Reference   Resolution
sfkclgateaccess.dll     %WINSYSDIR%\kernel32     incomplete  OS
                        %WINSYSDIR%\user32       incomplete  OS
                        %WINSYSDIR%\gdi32        incomplete  OS
                        %WINSYSDIR%\msvcrt       incomplete  OS
                        %INSTDIR%\sfkclgateslave.exe  unknown

Host module             Hosted/imported module   Reference   Resolution
sfkprotocolhdl.dll      %WINSYSDIR%\kernel32     incomplete  OS
                        %WINSYSDIR%\msvcrt       incomplete  OS
                           sfktools              incomplete  OS
                           sfbinet               incomplete  OS
                           sfbsolar              incomplete  OS
                           sfbtoolsn             incomplete  OS
```

Host module	Hosted/imported module	Reference	Resolution
sfktonac.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\gdi32	incomplete	OS
	%INSTDIR%\sfktonac.exe	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfktonlinedec.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfktools	incomplete	OS
	sfbsolar	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfbvos	incomplete	OS
	sfktonac	incomplete	OS
	sfkclgateaccess	incomplete	OS
	%INSTDIR%\CATClient\CLGate32.exe	unknown	
	versions.ini	unknown	
	passwort.ini	unknown	
	dbserver.ini	unknown	
	connect.ini	unknown	

Host module	Hosted/imported module	Reference	Resolution
sfkwebbankinghdl.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\ole32	incomplete	OS
	%WINSYSDIR%\oleaut32	incomplete	OS
	%WINSYSDIR%\wininet	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfkhbcihdl	incomplete	OS
	sfkprotocolhdl	incomplete	OS
	sfktools	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfbxmlutil	incomplete	OS
	sfbmsutil	incomplete	OS
	sfbvos	incomplete	OS
	sfkhbcitranslator	incomplete	OS
	sfkxsltscripthdl	incomplete	OS
	sfkwebengine	incomplete	OS
	sfbcore	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfkwebengine.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\oleaut32	incomplete	OS
	%WINSYSDIR%\wininet	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbcore	incomplete	OS
	sfbxmlutil	incomplete	OS
	sfbosl	incomplete	OS
	sfbvos	incomplete	OS
	sfbmsutil	incomplete	OS
	sfkwebenginecmd	incomplete	OS
	sfktools	incomplete	OS
	sfkxsltscripthdl	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfkwebenginecmd.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\oleaut32	incomplete	OS
	%WINSYSDIR%\wininet	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbcore	incomplete	OS
	sfbosl	incomplete	OS
	sfbvos	incomplete	OS
	sfbxmlutil	incomplete	OS
	sfkxsltscripthdl	incomplete	OS
	sfbmsutil	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfkxmlhttprequest	incomplete	OS
	sfktools	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfkxmlhttprequest.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\ole32	incomplete	OS
	%WINSYSDIR%\oleaut32	incomplete	OS
	%WINSYSDIR%\wininet	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbmsutil	incomplete	OS
	sfktools	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
CATClient\cptdll32.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\gdi32	incomplete	OS
	%WINSYSDIR%\shell32	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
	%WINSYSDIR%\comctl32	incomplete	OS
	%WINSYSDIR%\comdlg32	incomplete	OS
	%WINSYSDIR%\wsock32	incomplete	OS
	%WINSYSDIR%\version	incomplete	OS
	kitdll32	unknown	

Host module	Hosted/imported module	Reference	Resolution
CATClient\TrpS2032.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\gdi32	incomplete	OS

User interface 24 files

Host module	Hosted/imported module	Reference	Resolution
iecontr.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\ole32	incomplete	OS
	%WINSYSDIR%\gdi32	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
libtidy.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
olch2d32.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\gdi32	incomplete	OS
	%WINSYSDIR%\comdlg32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfbsv.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\ole32	incomplete	OS
	%WINSYSDIR%\gdi32	incomplete	OS
	%WINSYSDIR%\shell32	incomplete	OS
	%WINSYSDIR%\comdlg32	incomplete	OS
	%WINSYSDIR%\sage	incomplete	OS
	%WINSYSDIR%\winmm	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfbtoolsn	incomplete	OS
	sfbtoolso	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfbsvtools.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\gdi32	incomplete	OS
	%WINSYSDIR%\twain_32	incomplete	OS
	%WINSYSDIR%\version	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfbtoolsn	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbsv	incomplete	OS
	np*.dll	unknown	
	ipc.ini	unknown	
	sgf.ini	unknown	

Host module	Hosted/imported module	Reference	Resolution
sfmcorecobasic.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfkkernel	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfbunzip	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmcoremisc	incomplete	OS
	%WINDIR%\notepad.exe	unknown	
	%WINDIR%\hh.exe	unknown	
	shelp.exe	unknown	

Host module	Hosted/imported module	Reference	Resolution
sfmcorere.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsv	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmcoremisc	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmuicontrol.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\gdi32	incomplete	OS

	%WINSYSDIR%\shell32	incomplete	OS
	%WINSYSDIR%\riched20	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfkkernel	incomplete	OS
	sfbvos	incomplete	OS
	olch2d32	incomplete	OS
	sfmcorecoapp	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcorecosystemdb	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmcoreimex	incomplete	OS
	sfmcoremisc	incomplete	OS
	sfmuief	incomplete	OS
	sfmuihtmlhelp	incomplete	OS
	sfmuiie	incomplete	OS
	sfmuisvbrbx	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmuidetailaccount.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\shell32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfkkernel	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcorecostock	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmcoremisc	incomplete	OS
	sfmuicontrol	incomplete	OS
	sfmuidetailbasic	incomplete	OS
	sfmuisvbrbx	incomplete	OS
	sfmuiie	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmuidetailbasic.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbsdb	incomplete	OS

	sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfkkernel	incomplete	OS
	sfbvos	incomplete	OS
	sfmcorecoapp	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcorecostock	incomplete	OS
	sfmcorecosystemdb	incomplete	OS
	sfmcorecotimedep	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmcoremisc	incomplete	OS
	sfmuicontrol	incomplete	OS
	sfmuief	incomplete	OS
	sfmuiie	incomplete	OS
	sfmuisvbrbx	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmuidetailcommon.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbcore	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfkkernel	incomplete	OS
	sfbosl	incomplete	OS
	sfmcorecoapp	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmcoremisc	incomplete	OS
	sfmuicontrol	incomplete	OS
	sfmuidetailbasic	incomplete	OS
	sfmuisvbrbx	incomplete	OS
	sfmuiie	incomplete	OS
	sfmuihtmlhelp	incomplete	OS
	sfmcorecostock	incomplete	OS
	onlineup.ini	unknown	
	ouext.ini	unknown	

Host module	Hosted/imported module	Reference	Resolution
sfmuidetailhbauto.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
	%INSTDIR%\sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmuicontrol	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmuidetailoverview.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	olch2d32	incomplete	OS
	sfmcorecoapp	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcorecostock	incomplete	OS
	sfmcorecotimedep	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmcoremisc	incomplete	OS
	sfmuicontrol	incomplete	OS
	sfmuidetailbasic	incomplete	OS
	sfmuiie	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmuidetailreport.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcorecostock	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmcorere	incomplete	OS
	sfmuicontrol	incomplete	OS
	sfmuidetailbasic	incomplete	OS
	sfmuiie	incomplete	OS
	sfmuisvbrbx	incomplete	OS
	sfmcoremisc	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmuidetailstock.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfkkernel	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcorecostock	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmuicontrol	incomplete	OS
	sfmuidetailbasic	incomplete	OS
	sfmuisvbrbx	incomplete	OS
	sfmcoremisc	incomplete	OS
	sfmuiie	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmuidetailtimedep.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcorecotimedep	incomplete	OS
	sfmuicontrol	incomplete	OS
	sfmuidetailbasic	incomplete	OS
	sfmuisvbrbx	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmuiie	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmuidetailtransfer.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfkkernel	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcorecotimedep	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
	sfmcoredb	incomplete	OS
	sfmcoremisc	incomplete	OS
	sfmuicontrol	incomplete	OS
	sfmuidetailbasic	incomplete	OS
	sfmuisvbrbx	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmuief.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfbtoolsn	incomplete	OS
	sfbsv	incomplete	OS
	sfmcorecobasic	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmuihtmlhelp.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcoremisc	incomplete	OS
	%INSTDIR%\smoney.ini	unknown	

Host module	Hosted/imported module	Reference	Resolution
sfmuiie.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\ole32	incomplete	OS
	%WINSYSDIR%\oleaut32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfbsv	incomplete	OS
	sfmcorecobasic	incomplete	OS
	iecontr	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmuiimporterimpl.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	%INSTDIR%\smoney.ini	complete	Application

Host module	Hosted/imported module	Reference	Resolution
sfmuiimportersc2.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsv	incomplete	OS
	sfmuiimporterimpl	incomplete	OS
	sfmuiimportershare	incomplete	OS
	%INSTDIR%\smkonv.ini	complete	Application

Host module	Hosted/imported module	Reference	Resolution
sfmuiimportershare.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfbtoolsn	incomplete	OS
	sfbsv	incomplete	OS
	sfmuiimporterimpl	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmuiimporterzvlight.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsv	incomplete	OS
	sfmuiimportershare	incomplete	OS
	sfmuiimporterimpl	incomplete	OS
	omikron.ini	complete	Application

Miscellaneous 15 files

Host module	Hosted/imported module	Reference	Resolution
msvcp60.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
patchw32.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\ole32	incomplete	OS
	%WINSYSDIR%\shell32	incomplete	OS
	%WINSYSDIR%\mpr	incomplete	OS
	rtpmsi32	unknown	
	%WINSYSDIR%\version	incomplete	OS
	wininit.ini	complete	Application

Host module	Hosted/imported module	Reference	Resolution
sfbmsutil.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\ole32	incomplete	OS
	%WINSYSDIR%\oleaut32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolsn	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfbsolar.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\user32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfbtoolsn	incomplete	OS
	sfbvos	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfbtoolso.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\sfbtoolsn	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfbunzip.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfbversioninfo.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\version	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfbxmlutil.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\ole32	incomplete	OS
	%WINSYSDIR%\oleaut32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	libtidy	incomplete	OS
	sfbcore	incomplete	OS
	sfbmsutil	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfktools.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbcrypto	incomplete	OS
	sfbvos	incomplete	OS
	sfbsolar	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfbcore	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfkupdatehdl.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	patchw32	incomplete	OS
	sfktools	incomplete	OS
	sfbsolar	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfkprotocolhdl	incomplete	OS
	sfbinet	incomplete	OS
	sfbcore	incomplete	OS
	sfbversioninfo	incomplete	OS
	sfkupdat.ini	unknown	

Host module	Hosted/imported module	Reference	Resolution
sfkxinclude.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\ole32	incomplete	OS
	%WINSYSDIR%\oleaut32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbmsutil	incomplete	OS
	sfbxmlutil	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfkxsltscripthdl.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\shlwapi	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbcore	incomplete	OS
	sfbmsutil	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfkxinclude	incomplete	OS

Host module	Hosted/imported module	Reference	Resolution
sfmcoremisc.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\advapi32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbcore	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	%WINDIR%\win.ini	complete	Application

Host module	Hosted/imported module	Reference	Resolution
sfmuionlactiv.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\shell32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfktools	incomplete	OS
	sfbsvtools	incomplete	OS
	sfbsv	incomplete	OS
	sfmcorecoapp	incomplete	OS
	sfmcorecobasic	incomplete	OS
	sfmcoredb	incomplete	OS
	sfmcoremisc	incomplete	OS
	sfmuidetailbasic	incomplete	OS
	sfmuicontrol	incomplete	OS
	sfmuiie	incomplete	OS
	sfmuisvbrbx	incomplete	OS
	%IEDIR%\iexplore.exe	unknown	

Host module	Hosted/imported module	Reference	Resolution
sfmuisvbrbx.dll	%WINSYSDIR%\kernel32	incomplete	OS
	%WINSYSDIR%\msvcrt	incomplete	OS
	%INSTDIR%\msvcp60	incomplete	OS
	sfbtoolso	incomplete	OS
	sfbtoolsn	incomplete	OS
	sfbsv	incomplete	OS

References

[AAG95] G.D. Abowd, R. Allen, and D. Garlan. Formalizing style to understand descriptions of software architecture. *ACM Transactions on Software Engineering and Methodology*, 4(4):319–364, 1995.

[AC01] ACSA and The MITRE Corporation, editors. *Proceedings of ACSA Workshop on Information Security System Scoring and Ranking, Williamsburg, VA, U.S.A., May 21–23, 2001.* Applied Computer Security Associates, 2001.

[ASP03] N. Amálio, S. Stepney, and F. Polack. Modular UML semantics: Interpretation in Z based on templates and generics. In H.D. Van and Z. Liu, editors, *FACS'03 Workshop on Formal Aspects of Component Software*, volume 284 of *UNU/IIST Technical Report*, pages 81–100, 2003.

[Ass03] Computing Research Association. Grand research challenges in information systems. http://www.cra.org/reports/gc.systems.pdf, 2003.

[Boe81] B. Boehm. *Software Engineering Economics.* Prentice-Hall, 1981.

[BOLJ94] S. Brocklehurst, T. Olovsson, B. Littlewood, and E. Jonsson. On measurement of operational security. pdcs no. 160. Technical report, CEC ESPRIT Programme Basic Research Action Project 6362, PDCS 2 (Predictably Dependable Computing Systems 2), 1994.

[Bow96] J. Bowen. *Formal Specification and Documentation using Z: A Case Study Approach.* International Thomson Computer Press, 1996.

[BSC94] R. Barden, S. Stepney, and D. Cooper. *Z in Practice.* Prentice Hall, 1994.

[Bun05] Bundesnetzagentur. Einheitliche Spezifizierung der Einsatzbedingungen für Signaturanwendungskomponenten. Arbeitsgrundlage für Entwickler/Hersteller und Prüf-/Bestätigungsstellen, Version 1.4, 2005-07-19. http://www.bundesnetzagentur.de/media/archive/2648.pdf, 2005.

[BWG05] M.A. Babar, X. Wang, and I. Gorton. Supporting security sensitive architecture design. In R. Reussner, J. Mayer, J.A. Stafford, S. Overhage, S. Becker, and P.J. Schroeder, editors, *Quality of Software Architectures and Software Quality, First International Conference on the Quality of Software Architectures, QoSA 2005 and Second International Workshop on Software Quality, SOQUA 2005, Erfurt, Germany, September 20-22, 2005, LNCS 3712*, pages 140–154. Springer, 2005.

[CC299a] ISO 15408-1:1999, Evaluation criteria for IT security – part 1: Introduction and general model [Common Criteria for information technology security evaluation, version 2.1 part 1, "CC"], 1999.

[CC299b] ISO 15408-2:1999, Evaluation criteria for IT security – part 2: Security functional requirements, 1999.

[CC299c] ISO 15408-3:1999, Evaluation criteria for IT security – part 3: Security assurance requirements, 1999.

[CC305a] Common Criteria for information technology security evaluation, version 3.0, revision 2, June 2005. part 1: Introduction and general model, 2005.

[CC305b] Common Criteria for information technology security evaluation, version 3.0, revision 2, July 2005. part 2: Functional security components, 2005.

[CC305c] Common Criteria for information technology security evaluation, version 3.0, revision 2, July 2005. part 3: Security assurance components, 2005.

[CEM04] ISO 18045:2004, Methodology for IT security evaluation [Common Evaluation Methodology, version 2.2, "CEM"], 2004.

[CEM05] Common Methodology for information technology security evaluation, version 3.0, revision 2, July 2005, 2005.

[CH78a] A.B. Cremers and T.N. Hibbard. Formal modeling of virtual machines. *IEEE Transactions on Software Engineering*, 4(5):426–436, 1978.

[CH78b] A.B. Cremers and T.N. Hibbard. Functional behavior in data spaces. *Acta Informatica*, 9:293–307, 1978.

[Cle77] D.P. Clements. *Fuzzy Ratings for Computer Security Evaluation.* PhD thesis, University of California, Berkeley, 1977.

[CM97] P. Ciancarini and C. Mascolo. Analyzing and refining an architectural style. In J.P. Bowen, M.G. Hinchey, and D. Till, editors, *ZUM '97: The Z Formal Specification Notation, 10th International Conference of Z Users, LNCS 1212*, pages 349–368. Springer, 1997.

[Coh85] F. Cohen. *Computer Viruses.* PhD thesis, University of Southern California, 1985.

[Com00] Computer System Security and Privacy Advisory Board, editor. *Report of NIST CSSPAB workshop on security metrics, Gaithersburg, MD, U.S.A., June 13–14, 2000.* NIST, 2000.

[COR04] CORAS methodology. [modified 2004-10-17, down-loaded 2005-05-23]. http://coras.sourceforge.net/, 2004.

[CS00] D. Cooper and S. Stepney. Segregation with communication. In J.P. Bowen, S. Dunne, A. Galloway, and S. King, editors, *ZB 2000: Formal Specification and Development in Z and B, First International Conference of B and Z Users, LNCS 1878*, pages 451–470. Springer, 2000.

[CSL01] A.B. Cremers, A. Spalka, and H. Langweg. Vermeidung und Abwehr von Angriffen Trojanischer Pferde auf Digitale Signaturen. In Bundesamt für Sicherheit in der Informationstechnik, editor, *'2001 - Odyssee im Cyberspace? Sicherheit im Internet!' Tagungsband 7. Deutscher IT-Sicherheitskongress des BSI*, pages 113–125. SecuMedia Verlag, 2001.

[CTC93] *CTCPEC: The Canadian Trusted Computer Product Evaluation Criteria. Version 3.0e*. Canadian System Security Centre, 1993.

[Dac94] M. Dacier. *Vers une évaluation quantitative de la sécurité informatique*. PhD thesis, Institut National Polytechnique de Toulouse, 1994.

[DD94] M. Dacier and Y. Deswarte. Privilege graph: an extension to the typed access matrix model. In *Proceedings of the 1994 European Symposium on Research in Computer Security. LNCS 875*, pages 319–334. Springer, 1994.

[DGP+01] L. Davis, R. Gamble, J. Payton, G. Jónsdóttir, and D. Underwood. A notation for problematic architecture interactions. In *Proceedings of the 8th European software engineering conference*, pages 132–141. ACM, 2001.

[DKF+03] G. Denker, L. Kagal, T. Finin, M. Paolucci, and K. Sycara. Security for DAML web services: Annotation and matchmaking. In *Proceedings of 2nd International Semantic Web Conference, ISWC2003, LNCS 2870*, pages 335–350, 2003.

[Don03] M. Donner. Toward a security ontology. *IEEE Security & Privacy*, 1(3):6–7, 2003.

[Eng02] University of Southern California Center for Software Engineering. Cocomo. http://sunset.usc.edu/research/COCOMOII/index.html, 2002.

[Fin04] *FinTS Financial Transaction Services Version 4.0*. Bundesverband deutscher Banken e.V., Deutscher Sparkassen- und Giroverband e.V., Bundesverband der Deutschen Volksbanken und Raiffeisenbanken e.V., Bundesverband Öffentlicher Banken Deutschlands e.V., 2004.

[FP01] E.B. Fernandez and R. Pan. A pattern language for security models. In *8th Conference on Pattern Languages for Programs*, 2001.

[GAG05] G. Gousios, E. Aivaloglou, and S. Gritzalis. Distributed component architectures security issues. *Computer Standards & Interfaces*, 27(3):269–284, 2005.

[Gas88] M. Gasser. *Building a Secure Computer System*. Van Nostrand Reinhold, 1988.

[Gol05] D. Gollmann. *Computer Security*. John Wiley & Sons, second edition, 2005.

[Gra90] A. Gravell. What is a good formal specification? In J.E. Nicholls, editor, *Z User Workshop. Proceedings of the Fifth Annual Z User Meeting*, pages 137–150. Springer, 1990.

[Gre01] S.J. Greenwald. How i lost and then regained my faith in metrics. In *Position paper submitted to Workshop on Information-Security-System Rating and Ranking*. Applied Computer Security Associates (ACSA) and The MITRE Corporation, 2001.

[HBC00] *HBCI Homebanking Computer Interface Version 2.2*. Bundesverband deutscher Banken e.V., Deutscher Sparkassen- und Giroverband e.V., Bundesverband der Deutschen Volksbanken und Raiffeisenbanken e.V., Bundesverband Öffentlicher Banken Deutschlands e.V., 2000.

[HHA04] A. Hunstad, J. Hallberg, and R. Andersson. Measuring IT security – a method based on common criteria's security functional requirements. In *Proceedings of the 2004 IEEE Workshop on Information Assurance*, pages 226–233. IEEE Computer Society, 2004.

[HJA04] M. Hafiz, R.E. Johnson, and R. Afandi. The security architecture of qmail. In *11th Conference on Pattern Languages of Programs*, 2004.

[HK81] S.M. Henry and D. Kafura. Software structure metrics based on information flow. *IEEE Transactions on Software Engineering*, 7(SE):510–518, 1981. Reference given in [Kan02].

[HMC78] L.J. Hoffman, E.H. Michelman, and D. Clements. Securate – security evaluation and analysis using fuzzy metrics. In *Proceedings of 1978 National Computer Conference*, pages 531–540, 1978.

[Hog88] C.B. Hogan. Protection imperfect: The security of some computing environments. *Operating Systems Review*, 22(3):7–27, 1988.

[HPW03] M. Howard, J. Pincus, and J.M. Wing. Measuring relative attack surfaces. In *Proceedings of Workshop on Advanced Developments in Software and Systems Security, Taipei, Taiwan, December 2003*, 2003.

[HRU76] M.A. Harrison, W.L. Ruzzo, and J.D. Ullman. Protection in operating systems. *Communications of the ACM*, 19(8):461–471, 1976.

[IEE00] IEEE Recommended practice for architectural description of software-intensive systems. IEEE Standard 1471-2000, 2000.

[Int04] International Function Point Users Group. IFPUG web site, http://www.ifpug.org, 2004.

[ITS91] *Information Technology Security Evaluation Criteria (ITSEC). Version 1.2, 28.06.1991.* Commission of the European Communities, 1991.

[ITS93] *Information Technology Security Evaluation Manual (ITSEM). Version 1.0, 10.09.1993.* Commission of the European Communities, 1993.

[Jac97] J. Jacky. *The way of Z.* Cambridge University Press, 1997.

[Kan02] S.H. Kan. *Metrics and Models in Software Quality Engineering.* Addison–Wesley, second edition, 2002.

[KC00] S. Kim and D. Carrington. A formal mapping between UML models and Object-Z specifications. In J.P. Bowen, S. Dunne, A. Galloway, and S. King, editors, *ZB 2000: Formal Specification and Development in Z and B, First International Conference of B and Z Users, LNCS 1878*, pages 2–21. Springer, 2000.

[KF05] B. Krüger and D. Feldhusen (SRC Security Research and Consulting GmbH). Personal communication, 2005.

[LABMC94] C.E. Landwehr, A.R. A.R. Bull, J.P. McDermott, and W.S. Choi. A taxonomy of computer program security flaws. *ACM Computing Surveys*, 26(3):211–254, 1994.

[Lan02] H. Langweg. With gaming technology towards secure user interfaces. In *Proceedings of 18th Annual Computer Security Applications Conference*, pages 44–50. IEEE Computer Society, 2002.

[Lan03a] H. Langweg. Sichere Benutzeroberflächen mittels Spieletechnologie. Presented at 8. Deutscher IT-Sicherheitskongress des BSI, 2003.

[Lan03b] H. Langweg. Sicherere Benutzeroberflächen mittels DirectX. In R. Grimm, H.B. Keller, and K. Rannenberg, editors, *'Informatik 2003. Mit Sicherheit Informatik.' Tagungsband Jahrestagung der Gesellschaft für Informatik, Schwerpunkt 'Sicherheit - Schutz und Zuverlässigkeit'*, pages 227–236. Gesellschaft für Informatik, 2003.

[Lan04a] H. Langweg. Building a trusted path for applications using COTS components. In *Proceedings of NATO RTO IST Panel Symposium on Adaptive Defence in Unclassified Networks*, pages 21–1–21–14, 2004.

[Lan04b] H. Langweg. If you stretch it too far, it breaks – challenges of biased technology. In P. Duquenoy, S. Fischer-Hübner, J. Holvast, and A. Zuccato, editors, *Risks and Challenges of the Network Society – Proceedings of the Second IFIP 9.2, 9.6/11.7 Summer School 4–8 August 2003*, volume 2004:35 of *Karlstad University Studies*, pages 236–241, 2004.

[Lan05] H. Langweg. Eine CC-basierte Ontologie zur Analyse architektureller Schwachstellen. Presented at 9. Deutscher IT-Sicherheitskongress des BSI, 2005.

[Lan06a] H. Langweg. Framework for malware resistance metrics. In *Proceedings of 13th ACM Conference on Computer and Communications Security, Second Workshop on Quality of Protection, Alexandria, VA, U.S.A., October 30th, 2006*, pages 39–44. ACM, 2006.

[Lan06b] H. Langweg. Malware attacks on electronic signatures revisited. In J. Dittmann, editor, *'Sicherheit 2006'. Konferenzband der 3. Jahrestagung Fachbereich Sicherheit der Gesellschaft für Informatik.*, pages 244–255. Gesellschaft für Informatik, 2006.

[Lei00] F. Leitold. Mathematical model of computer viruses. In U.E. Gattiker, editor, *EICAR 2000 Best Paper Proceedings, Annual Meeting of European Institute for Computer Antivirus Research, Brussels, Belgium, March 4-7, 2000*, pages 194–217, 2000.

[LG99] P.T.L. Lloyd and G.M. Galambos. Technical reference architectures. *IBM Systems Journal*, 38(1):51–75, 1999.

[LK06] H. Langweg and T. Kristiansen. Securing the weak link in client-server interaction. In *arXiv Computing Research Repository, cs.CR/0611102*, 2006.

[LS04] H. Langweg and E. Snekkenes. A classification of malicious software attacks. In *Proceedings of 23rd IEEE International Performance, Computing, and Communications Conference*, pages 827–832. IEEE Computer Society, 2004.

[LS07] H. Langweg and J. Schwenk. Schutz von FinTS/HBCI-Clients gegenüber Malware. In P. Horster, editor, *Proceedings of D-A-CH Security*, pages 227–238, 2007.

[McC76] T.J. McCabe. A complexity measure. *IEEE Transactions on Software Engineering*, 2(4):308–320, 1976. Reference given in [She95].

[MCC84] G.E. Murine and Jr. C.L. Carpenter. Measuring computer system security using software security metrics. In J.H. Finch and E.G. Dougall, editors, *Computer Security: A Global Challenge, Proceedings of the Second IFIP International Conference on Computer Security, IFIP/SEC'84, Toronto, Ontario, Canada, September 10-12, 1984*, pages 207–215. Elsevier, 1984.

[Met06] *Proceedings of 1st Workshop on Security Metrics (MetriCon 1.0), affiliated with 15th USENIX Security Symposium, Vancouver, BC, Canada, July 31st, 2006.* USENIX, 2006.

[MGM03] H. Mouratidis, P. Giorgini, and G. Manson. An ontology for modelling security: The tropos approach. In *Proceedings of Knowledge-Based Intelligent Information and Engineering Systems: 7th International Conference, KES 2003, LNCS 2773*, pages 1387–1394, 2003.

[MP93] K.H. Möller and D.J. Paulish. *Software Metrics. A Practitioner's Guide to Improved Product Development.* Chapman & Hall, 1993.

[MS90] J.D. Moffett and M.S. Sloman. A case study in representing a model: To Z or not to Z. In J.E. Nicholls, editor, *Z User Workshop. Proceedings of the Fifth Annual Z User Meeting*, pages 254–268. Springer, 1990.

[MvOV96] A.J. Menezes, P.C. van Oorschot, and S.A. Vanstone. *Handbook of Applied Cryptography.* CRC Press, 1996.

[MY06] S. Myagmar and W. Yurcik. Why johnny can hack: The mismatch between vulnerabilities and security protection standards. In *IEEE International Symposium on Secure Software Engineering (ISSSE), McLean, VA, U.S.A.*, 2006.

[Mye80] P. Myers. *Subversion: The Neglected Aspect of Computer Security.* MSc thesis, Naval Postgraduate School, 1980.

[NB05] T. Nakamura and V.R. Basili. Metrics of software architecture changes based on structural distance. In *Proceedings of the 11th IEEE International Software Metrics Symposium (METRICS 2005)*, pages 8.1–8.10. IEEE, 2005.

[Neu96] P.G. Neumann. Architectures and formal representations for secure systems. final report sri project 6401 deliverable a002. Technical report, SRI International, 1996.

[Neu00] P.G. Neumann. Practical architectures for survivable systems and networks (phase-two final report). Technical report, SRI International, 2000.

[NIS03] *Security Metrics Guide for Information Technology Systems. NIST Special Publication 800-55*, 2003.

[NJOJ03] S. Noel, S. Jajodia, B. O'Berry, and M. Jacobs. Efficient minimum-cost network hardening via exploit dependency graphs. In *Proceedings of 19th Annual Computer Security Applications Conference*, pages 86–95. IEEE Computer Society, 2003.

[PST96] B. Potter, J. Sinclair, and D. Till. *An Introduction to Formal Specification and Z.* Prentice Hall, second edition, 1996.

[QoP05] *Proceedings of 1st QoP Workshop on Quality of Protection, affiliated with 10th European Symposium of Research in Computer Security and 11th IEEE International Software Metrics Symposium, Milano, Italy, September 15th, 2005.* Kluwer, 2005.

[QoP06] *Proceedings of 2nd QoP Workshop on Quality of Protection, affiliated with 13th ACM Conference on Computer and Communications Security, Alexandria, VA, U.S.A., October 30th, 2006.* ACM, 2006.

[Res04] E. Rescorla. Is finding security holes a good idea? In *The Third Annual Workshop on Economics and Information Security (WEIS04)*, 2004.

[Rom90] H.D. Rombach. Design measurement: Some lessons learned. *IEEE Software*, 7(2):17–25, 1990.

[SC96] M. Shaw and P. Clements. Toward boxology: Preliminary classification of architectural styles. In *Proceedings of Second International Software Architecture Workshop (ISAW-2)*, pages 50–54. ACM, 1996.

[Sch99] E.A. Schneider. Security architecture-based system design. In *Proceedings of New Security Paradigms Workshop 1999*, pages 25–31, 1999.

[SCH04] A. Sachitano, R.O. Chapman, and J.A. Hamilton. Security in software architecture: A case study. In *Proceedings of the 2004 IEEE Workshop on Information Assurance*, pages 370–376. IEEE Computer Society, 2004.

[SCL00] A. Spalka, A.B. Cremers, and H. Lehmler. Protecting confidentiality against trojan horse programs in discretionary access control system. In *Proceedings of the 5th Australasian Conference on Information Security and Privacy, LNCS 1841*, pages 1–17. Springer, 2000.

[SCL01a] A. Spalka, A.B. Cremers, and H. Langweg. The fairy tale of 'what you see is what you sign' - trojan horse attacks on software for digital signatures. In S. Fischer-Hübner, D. Olejar, and K. Rannenberg, editors, *Security & Control of IT in Society – II (SCITS–II). Proceedings of the IFIP WG 9.6/11.7 Working Conference*, pages 75–86, 2001.

[SCL01b] A. Spalka, A.B. Cremers, and H. Langweg. Protecting the creation of digital signatures with trusted computing platform technology against attacks by trojan horse programs. In M. Dupuy and P. Paradinas, editors, *Trusted Information. The New Decade Challenge. Proceedings of IFIP/SEC'01*, pages 403–419. Kluwer, 2001.

[SCL02] A. Spalka, A.B. Cremers, and H. Langweg. Trojan horse attacks on software for electronic signatures. *Informatica, Special Issue 'Security and Protection'*, 26(2):191–204, 2002.

[She95] M. Shepperd. *Foundations of software measurement*. Prentice Hall, 1995.

[SHJ+02] O. Sheyner, J. Haines, S. Jha, R. Lippmann, and J. Wing. Automated generation and analysis of attack graphs. In *Proceedings of the 2002 IEEE Symposium on Security and Privacy*, pages 273–284. IEEE Computer Society, 2002.

[Sim05] J. Simpson, editor. *The Oxford English Dictionary*. Oxford University Press, online edition, 2005.

[SL02a] A. Spalka and H. Langweg. Notes on program-orientated access control. In *Proceedings of International Workshop on Trust and Privacy in Digital Business (TrustBus 2002) held in conjunction with 13th International Workshop on Database and Expert Systems Applications (DEXA 2002)*, pages 451–455. IEEE Computer Society, 2002.

[SL02b] A. Spalka and H. Langweg. Protecting the user from the data: Security and privacy aspects of public web access. In *Proceedings of 2nd International Conference on Adaptive Hypermedia and Adaptive Web Based Systems (AH'02), LNCS 2347*, pages 440–443. Springer, 2002.

[SO01] G. Sindre and A.L. Opdahl. Templates for misuse case description. In *7th International Workshop on Requirements Engineering, Foundation for Software Quality (REFSQ 2001)*, 2001.

[Spi92] J.M. Spivey. *The Z Notation: A Reference Manual*. Prentice Hall, second edition, 1992.

[SS75] J.H. Saltzer and M.D. Schroeder. The protection of information in computer systems. *Proceedings of the IEEE*, 63(9):1278–1308, 1975.

[SSE03] *Systems Security Engineering Capability Maturity Model (SSE-CMM), Model Description Document, Version 3.0*. SSE-CMM Project, 2003.

[SSM97] K.J. Sullivan, J. Socha, and M. Marchukov. Using formal methods to reason about architectural standards. In *Proceedings of the 19th international conference on Software engineering*, pages 503–513. ACM, 1997.

[Sti00] O. Stiemerling. *Component-Based Tailorability*. PhD thesis, Rheinische Friedrich-Wilhelms-Universität Bonn, 2000.

[SW00] G. Schudel and B. Wood. Adversary work factor as a metric for information assurance. In *Proceedings of New Security Paradigms Workshop 2000*, pages 23–30. ACM, 2000.

[TCS85] *TCSEC: DoD 5200.28-STD Department of Defense Trusted Computer System Evaluation Criteria*. Department of Defense, 1985.

[vABHL03] L. von Ahn, M. Blum, N. Hopper, and J. Langford. CAPTCHA: Telling humans and computers apart. In E. Biham, editor, *Advances in Cryptology – EUROCRYPT 2003: International Conference on the Theory and Applications of Cryptographic Techniques, Warsaw, Poland, May 4-8, 2003, LNCS 2656*, pages 294–311. Springer, 2003.

[WBG+87] C.C. Wood, W.W. Banks, S.B. Guarro, V.E. Hampel, and H.P. Sartorio. *Computer security: a comprehensive controls checklist*. J. Wiley & Sons, 1987.

[WCLS02] M. Winandy, A.B. Cremers, H. Langweg, and A. Spalka. Protecting java component integrity against trojan horse programs. In M. Gertz, editor, *Proceedings of Integrity and Internal Control in Information Systems (IICIS 2002)*, pages 99–113. Kluwer, 2002.

[Whi01] J.J. Whitmore. A method for designing secure solutions. *IBM Systems Journal*, 40(3):747–768, 2001.

[YJB97] J. Yoder and J. J. Barcalow. Architectural patterns for enabling application security. In *4th Pattern Languages of Programming Conference, Washington University Technical Report 97-34*, 1997.

[Z00] ISO 13568:2000, Formal specification – Z notation – syntax, type and semantics, 2000.

[Zen89] Zentralstelle für Sicherheit in der Informationstechnik. Kriterien für die Bewertung der Sicherheit von Systemen der Informationstechnik (IT) – IT-Sicherheitskriterien – 1. Fassung vom 11.01.1989. In *GMBl [Gemeinsames Ministerialblatt]*, volume 40, pages 278–390. Bundesminister des Innern, 1989.

[Zen90] Zentralstelle für Sicherheit in der Informationstechnik. Handbuch für die Prüfung der Sicherheit von Systemen der Informationstechnik (IT) – IT-Evaluationshandbuch – 1. Fassung vom 22.02.1990. In *GMBl [Gemeinsames Ministerialblatt]*, volume 41, pages 430–536. Bundesminister des Innern, 1990.

Index

Architecture, 45, 53, 173, 174, 183
Attacker capabilities, 48, 160
  Attack initiation, 48, 49
  Attack variation, 48, 49
  Available time to attack, 48, 49
  Influence on user, 49
  Level, 50
Attacks
  Generic, 50, 127
  Repository, 51, 127

Common Criteria, 37, 71
Component type
  Access control data, 75
  Data, 78, 80, 82
  Firmware, 80
  Local human user, 82
  Logging configuration data, 75
  Reference, 78
  Subject–Adversary, 75
  Subject–Operating System, 75
  Subject–Unspecified, 75
  Subject–Victim, 75
  Tamper-proof storage, 78, 80
  UI Input, 82
  UI Output, 82
Connector type
  Access–append, 77
  Access–delete, 77
  Access–invoke, 77
  Access–modify, 77
  Access–observe, 76
  Backup, 79
  Contained–by, 79
  Data transfer, 79
  Execute, 81, 82
  Integrity verification data, 76
  Linked, 81
  Log data, 76, 82
  Monitor, 76, 82
  Ownership, 76
  Parameter, 76
  Reference rule–Container, 79
  Reference rule–Search order, 79
  Reference rule–Static, 79
  Security parameter, 76, 82
  Subject binding, 81
CORAS, 41, 73, 199

Evaluation, 45

Formal model, 87
  Architecture, 88, 100
  Given sets, 88
  Global constants, 88
  Initial state, 101
  Operations, 108
  State, 92

Generic attacks
  Add stored code module, 138
  Initiate communication and send data, 144
  Modify code in memory, 134
  Modify reference to stored code module, 141
  Modify reference to stored data component containing parameters, 151
  Modify stored code module, 136
  Modify stored data component, 128
  Modify stored data component used for decisions, 157
  Modify stored parameters, 149
  Modify user interface object, 155
  Observe contents of stored data component, 132
  Respond to communication and send data, 147
  Simulate user input, 153

Metrics, 46, 48, 57, 69, 175
  Conformity of access permissions, 63

Percentage of access control instrumentation, 63
Percentage of authenticity/integrity preserving connectors, 64
Percentage of logged invocations, 64
Percentage of protected executables, 62
Percentage of protected intermediate storage components, 62
Percentage of trusted path connectors, 67
Percentage of unlogged security parameters, 65
Restriction of number of components with multiple executable extensions, 66
Restriction of number of components with shared responsibility (server), 65
Restriction of number of executable components, 61
Restriction of number of executable distribution sources, 60
Restriction of number of privileges, 68
Restriction of number of processes sharing a privilege, 68

Resistance class, 52, 162, 181
Role
Authenticated data source, 80, 81
Authenticated data target, 80, 81
Contained, 80
Container, 80
Copy, 79
Data source, 80, 81
Data target, 80, 81
Executed, 81
Executor, 81
Link executed dynamic, 81
Link executed static, 81
Link executor, 81
Log data source, 78
Log data target, 78
Monitor source, 78
Monitor target, 78
Object, 78
Original, 77, 79
Owned, 78
Owner, 78
Parameter processor, 77
Reference source, 80
Reference target, 80
Security data source, 77
Subject, 78
Verification data, 77

Security evaluation, *see* Evaluation
Security requirements, 46, 159
Code integrity, 46, 47
Data confidentiality, 46, 47
Data integrity, 46, 47
Level, 47

Curriculum vitae

01/2007–present	eQ-3 Entwicklung GmbH, Leer; Product Development *Time and Attendance*, *Physical Access Control*
10/2003–12/2006	PhD scholarship, Høgskolen i Gjøvik, Norway
01/2002–present	PhD student, Rheinische Friedrich-Wilhelms-Universität Bonn
01/2002–09/2003	Research Assistant, Department of Computer Science III, Rheinische Friedrich-Wilhelms-Universität Bonn
10/1995–12/2001	Studies of Computer Science (Diplom-Informatiker), Physical Geography, Law, and Norwegian at Rheinische Friedrich-Wilhelms-Universität Bonn

List of peer-reviewed publications

Articles in journals

1. A. Spalka, A.B. Cremers, and H. Langweg. Trojan horse attacks on software for electronic signatures. *Informatica, Special Issue 'Security and Protection'*, 26(2):191–204, 2002

Articles in peer-reviewed conference proceedings

1. H. Langweg and J. Schwenk. Schutz von FinTS/HBCI-Clients gegenüber Malware. In P. Horster, editor, *Proceedings of D-A-CH Security*, pages 227–238, 2007

2. H. Langweg. Framework for malware resistance metrics. In *Proceedings of 13th ACM Conference on Computer and Communications Security, Second Workshop on Quality of Protection, Alexandria, VA, U.S.A., October 30th, 2006*, pages 39–44. ACM, 2006

3. H. Langweg. Malware attacks on electronic signatures revisited. In J. Dittmann, editor, *'Sicherheit 2006'. Konferenzband der 3. Jahrestagung Fachbereich Sicherheit der Gesellschaft für Informatik.*, pages 244–255. Gesellschaft für Informatik, 2006

4. H. Langweg. Building a trusted path for applications using COTS components. In *Proceedings of NATO RTO IST Panel Symposium on Adaptive Defence in Unclassified Networks*, pages 21–1–21–14, 2004

5. H. Langweg and E. Snekkenes. A classification of malicious software attacks. In *Proceedings of 23rd IEEE International Performance, Computing, and Communications Conference*, pages 827–832. IEEE Computer Society, 2004

6. H. Langweg. If you stretch it too far, it breaks – challenges of biased technology. In P. Duquenoy, S. Fischer-Hübner, J. Holvast, and A. Zuccato, editors, *Risks and Challenges of the Network Society – Proceedings of the Second IFIP 9.2, 9.6/11.7 Summer School 4–8 August 2003*, volume 2004:35 of *Karlstad University Studies*, pages 236–241, 2004

7. H. Langweg. Sicherere Benutzeroberflächen mittels DirectX. In R. Grimm, H.B. Keller, and K. Rannenberg, editors, *'Informatik 2003. Mit Sicherheit Informatik.' Tagungsband Jahrestagung der Gesellschaft für Informatik, Schwerpunkt 'Sicherheit - Schutz und Zuverlässigkeit'*, pages 227–236. Gesellschaft für Informatik, 2003

8. H. Langweg. With gaming technology towards secure user interfaces. In *Proceedings of 18th Annual Computer Security Applications Conference*, pages 44–50. IEEE Computer Society, 2002

9. M. Winandy, A.B. Cremers, H. Langweg, and A. Spalka. Protecting java component integrity against trojan horse programs. In M. Gertz, editor, *Proceedings of Integrity and Internal Control in Information Systems (IICIS 2002)*, pages 99–113. Kluwer, 2002

10. A. Spalka and H. Langweg. Notes on program-orientated access control. In *Proceedings of International Workshop on Trust and Privacy in Digital Business (TrustBus 2002) held in conjunction with 13th International Workshop on Database and Expert Systems Applications (DEXA 2002)*, pages 451–455. IEEE Computer Society, 2002

11. A. Spalka and H. Langweg. Protecting the user from the data: Security and privacy aspects of public web access. In *Proceedings of 2nd International Conference on Adaptive Hypermedia and Adaptive Web Based Systems (AH'02), LNCS 2347*, pages 440–443. Springer, 2002

12. A. Spalka, A.B. Cremers, and H. Langweg. Protecting the creation of digital signatures with trusted computing platform technology against attacks by trojan horse programs. In M. Dupuy and P. Paradinas, editors, *Trusted Information. The New Decade Challenge. Proceedings of IFIP/SEC'01*, pages 403–419. Kluwer, 2001

13. A. Spalka, A.B. Cremers, and H. Langweg. The fairy tale of 'what you see is what you sign' - trojan horse attacks on software for digital signatures. In S. Fischer-Hübner, D. Olejar, and K. Rannenberg, editors, *Security & Control of IT in Society – II (SCITS–II). Proceedings of the IFIP WG 9.6/11.7 Working Conference*, pages 75–86, 2001

14. A.B. Cremers, A. Spalka, and H. Langweg. Vermeidung und Abwehr von Angriffen Trojanischer Pferde auf Digitale Signaturen. In Bundesamt für Sicherheit in der Informationstechnik, editor, *'2001 - Odyssee im Cyberspace? Sicherheit im Internet!' Tagungsband 7. Deutscher IT-Sicherheitskongress des BSI*, pages 113–125. SecuMedia Verlag, 2001

Poster presentations

1. H. Langweg. Eine CC-basierte Ontologie zur Analyse architektureller Schwachstellen. Presented at 9. Deutscher IT-Sicherheitskongress des BSI, 2005

2. H. Langweg. Sichere Benutzeroberflächen mittels Spieletechnologie. Presented at 8. Deutscher IT-Sicherheitskongress des BSI, 2003

Technical reports or archival repositories without peer review

1. H. Langweg and T. Kristiansen. Securing the weak link in client-server interaction. In *arXiv Computing Research Repository, cs.CR/0611102*, 2006