Detection of Unknown Cyber Attacks Using Convolution Kernels Over Attributed Language Models

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

von
Patrick Duessel
aus
Berlin

Bonn, Juli 2018

Dieser Forschungsbericht wurde als Dissertation von der Mathematisch-Naturwissenschaftlichen Fakultät der Universität Bonn angenommen und ist auf dem Hochschulschriftenserver der ULB Bonn http://hss.ulb.uni-bonn.de/diss_online elektronisch publiziert.

1. Gutachter:	Prof. Dr. Michael Meier
2. Gutachter:	Prof. Dr. Sven Dietrich
Tag der Promotion:	2. Juli 2018
Erscheinungsjahr:	2018

To my beloved family.

Acknowledgements

I would like to express my gratitude to my supervisor Prof. Dr. Michael Meier for continuous support and advise throughout my dissertation journey. I would also like to thank Prof. Dr. Sven Dietrich for providing direction to my thesis as co-referent. Furthermore, I would like to thank Prof. Dr. Klaus-Robert Mueller for his scientific inspiration and his support to foster my interest for research and development in the field of machine learning and network security during the first years of my studies.

I would also like to express my gratitude to all my colleagues at the Fraunhofer Institute, Berlin Institute of Technology as well as University of Bonn for fruitful discussions as well as an interactive and collaborative working atmosphere. I would like to thank my friends and colleagues Christian Gehl, Dr. Ulrich Flegel and Dr. Alexandra James for great moments shared and their personal support. Finally, I would like to express my love and gratitude to my family for their persistent moral support and for everything they have done for me.

Contents

CHAPTER 1

Introduction

Information Technology (IT) plays a crucial role in modern society. Over the past decade the Internet has emerged as a key communication platform for businesses and private users demonstrated by an exponential increase not only of hosts connected to the Internet but also Internet users. While in 2007 561.6 Million hosts were advertised in the Domain Name System (DNS) [1], the number of hosts has almost doubled to 1.062 Billion [2] over the past 10 years [1]. In addition, with the ongoing adoption of Internet of Things (IoT) as key technology enabler for advanced infrastructure concepts such as smart cities, smart grids, virtual power plants, or intelligent transportation systems (e.g. connected cars), the number of hosts connected to the Internet is expected to increase exponentially to 30 Billion devices by 2020 [2]. With an increase of hosts on the Internet, the number of Internet users has also increased dramatically over the past decade. While in 2007 20% of the world's population (1,319 Billion) utilized the Internet, the number of Internet users in 2017 has more than doubled to 3,885 Billion users (approx. 52% of the world's population [3]).

Considering the pervasiveness of today's IT organizations become increasingly exposed – predominantly driven by threat likelihood and vulnerability level of the organization. On the one hand, the number of reported security incidents world-wide has increased tremendously at a compound annual growth rate of approx. 60% between 2009 (3.4 Million incidents) and 2015 (59 Million incidents) [4]. One reason for the tremendous growth of computer and network attacks is an increase of attack automation and attack sophistication [5]. While in earlier days computer attacks had to be crafted manually and were aimed at specific targets, today complex malware tool kits are readily available to infiltrate a network, persistently deploy on target hosts, selectively propagate across the network and perform a wide range of actions ranging from sensitive data ex-filtration to distributed denial of service attacks [6]. A common concept to mount sophisticated and large-scale attacks are *Botnets* in which hosts are infected by malware to receive commands from a Command & Control (C2) server to execute malicious payload [7]. For example, the self-propagating worm *Mirai* infected over 600.000 IoT devices (mostly cameras and routers). Infected devices were controlled by so called C2 servers to mount the largest distributed denial of service attack on record with more than one Terabits per second (Tbps) at its peak causing temporary but significant connectivity issues for Internet users.

On the other hand, while there is an increase in threats, the vulnerability surface of organizations is also growing. As an example, the number of disclosed software vulnerabilities has grown at a compound annual growth rate of approx. 8.3% from 6,516 in 2007 to 14,451 reported vulnerabilities in 2017 [8]. Lack of security awareness, continuous adoption of emerging technology, increasingly complex software

[1] Devices behind Network Address Translation (NAT) segmented networks are not included in the estimation.
[2] Billion = 10^9

combined with fast-paced software development life cycles can be considered major factors that drive an organization's vulnerability level [9].

The following sections will provide an overview of challenges in the fields of computer and network security – specifically related to the detection of network-based computer attacks and host-based malware attacks – and furthermore introduce a machine learning-based approach to overcome these challenges.

1.1 Known and unknown threats

One of today's key challenges is the detection of unknown threats. The majority of the state-of-the-art security controls deployed in organizations are reactive in nature and predominantly based on known attack signatures. Although those safeguards demonstrate high detection accuracy for known attack patterns, drawbacks include their inability to reliably detect not only unknown but also modified versions of known threats. For example, in the context of targeted attacks by adversaries with advanced capabilities (e.g. state-sponsored attackers), computer and network attacks have become increasingly sophisticated. Obfuscation techniques such as polymorphism, metamorphism, armoring or simply using "living-off-the-land" functionality (i.e. file-less malware) are specifically designed to thwart existing countermeasures and enable undetected delivery and execution of malicious code [e.g. 10, 11].

The problem of unknown threat detection is known in many different areas in computer and network security. Network attacks and malware are among the most common threat vectors [12] and thus, are focal point of this dissertation.

Network attacks. Extensive research has been conducted to develop methods and systems for the detection of computer and network attacks. A computer attack refers to an attempt to compromise confidentiality, availability or integrity of a computing resource. An Intrusion Detection System (IDS) captures and analyzes streams of data in networks or on hosts to detect computer attacks. The conceptual design of intrusion detection systems goes back to the work of Denning [13] which serves as a foundation for numerous state-of-the-art intrusion detection systems nowadays [e.g. 14, 15]. An overview of existing intrusion detection approaches is provided by the work of McHugh [5]. Intrusion detection techniques can be broadly classified into misuse detection and anomaly detection [16]. While misuse detection methods are intended to recognize known attack patterns, anomaly detection techniques aim at identifying unusual structural or activity patterns in observed data. A well-known drawback of misuse detection systems is their inability to detect "unknown" attacks due to their reliance on pre-defined policies or attack signatures. Thus, significant research has been conducted on anomaly-based detection methods [e.g. 17–19]. However, the majority of these methods focuses on spotting computer and network attacks exploiting network protocol header vulnerabilities such as *Teardrop*, *Land* or *Ping of Death* [20].

Nowadays, the vast majority of network attacks is carried out at the application layer exploiting vulnerabilities in applications or third party software components by leveraging standard network protocols to deliver malicious exploit payload. Driven by the growing automation of vulnerability research and exploit development tools and techniques [e.g. 21, 22], exploit development cycles become increasingly shorter, leading to faster production and proliferation of commercial-grade software exploits. A *zero-day* vulnerability refers to a software or hardware flaw that is unknown to the security community and no fix or patch is available [23]. A *zero-day* attack refers to an attack exploiting a zero-day vulnerability whereas the time between discovery of the vulnerability and its exploitation is less than a day. For example, a recent attack launched against a major U.S. credit bureau showed how quickly a *zero-day* vulnerability can be exploited [24]. In that particular example, an exploit was developed and executed against a server containing an unfixed vulnerability (CVE-2017-5638 [25]) within less than 24

hours. The example below shows a Hyper-Text Transfer Protocol (HTTP) [26] request carrying malicious payload to exploit the aforementioned vulnerability residing in the Jakarta-based file upload multi-part parser of the *Apache Struts 2* web application framework [27] installed on one of the servers. In the vulnerable version (i.e. 2.3.2 or lower), incorrect exception handling and error message generation during file upload allow for arbitrary remote code execution via command injection in the *Content-Type* field of the HTTP request. In the example below, the current directory of the server is read out (i.e.#cmd=ḋir) and returned as output in the HTTP response. More sophisticated variants of that exploit have surfaced which include reconfiguration of security settings or utilization of persistency mechanisms [28].

```
GET / HTTP/1.1
Host: 192.168.126.128:80
User-Agent: Mozilla/5.0
Accept-Encoding: identity
Content-Type: %{(#_='multipart/form-data').(#dm=@ognl.
    OgnlContext@DEFAULT_MEMBER_ACCESS).(#_memberAccess?(#_memberAccess=#dm):((#
    container=#context['com.opensymphony.xwork2.ActionContext.container']).(#ognlUtil
    =#container.getInstance(@com.opensymphony.xwork2.ognl.OgnlUtil@class)).(#ognlUtil.
    getExcludedPackageNames().clear()).(#ognlUtil.getExcludedClasses().clear()).(#
    context.setMemberAccess(#dm)))).(#cmd='dir').(#iswin=(@java.lang.
    System@getProperty('os.name').toLowerCase().contains('win'))).(#cmds=(#iswin?{'cmd
    .exe','/c',#cmd}:{'/bin/bash','-c',#cmd})).(#p=new java.lang.ProcessBuilder(#cmds)
    ).(#p.redirectErrorStream(true)).(#process=#p.start()).(#ros=(@org.apache.struts2.
    ServletActionContext@getResponse().getOutputStream())).(@org.apache.commons.io.
    IOUtils@copy(#process.getInputStream(),#ros)).(#ros.flush())}
Connection: close
```

Listing 1.1: Malicious HTTP request exploiting vulnerability in the Apache Struts framework (CVE-2017-5638)

Although the HTTP request shows distinctive characteristics of malicious or at least suspicious payload (i.e. HTTP Content-Type parameter value), exploit authors can leverage various techniques (e.g. utilization of different character encodings, encryption etc.) to obfuscate the malicious payload, rendering traditional misuse detection ineffective to detect malicious payload delivered over application-level network protocols.

Advanced malware. The continuous proliferation of increasingly sophisticated malware poses a significant threat to organizations. Malware detection methods can be broadly categorized into *static* and *dynamic* analysis. While static analysis refers to the decompilation and analysis of structure, flow and data residing within the malicious binary at compile time [e.g. 29–32], dynamic analysis aims at profiling activity of malware binaries at run time [e.g. 33–37]. A major drawback of existing methods is its reliance on malware signatures resulting in limited ability to detect unknown malware or variants of known malware. This is reflected in detection accuracies between 18 to 70% – demonstrating a large variance between operating systems [38].
Nowadays, staging and obfuscation techniques are oftentimes used by malware authors in an attempt to evade anti-malware solutions. Staging refers to the segregation of malware functionality into download and execution stages, primarily to reduce the overall size of the malware during installation and hide malicious activity in normal network traffic during downloading of malicious payload. Multi-stage malware is a crucial component of today's *Botnets*, i.e. networks of compromised machines (i.e. bots) controlled by attackers through a C2 server infrastructure [39]. As opposed to single-stage malware in which malicious code is directly contained in the *Dropper*, multi-stage malware first downloads to

the victim's machine (e.g. via phishing or DNS hijacking) to establish persistency and consequently execute malicious payload obtained from a C2 server. Obfuscation is another technique used by malware authors to hide malicious characteristics of malware. Techniques such as *oligomorphism*, *polymorphism* or *metamorphism* aim at changing the representation (e.g. function call obfuscation, control flow obfuscation behavior) of malicious code while maintaining its functionality [e.g. 40].
IoT is as an example for growing sophistication and complexity of malware. The number of malware-enabled attacks on IoT devices has significantly increased by approx. 600% in 2017 (6.000 attacks in 2016). The most common IoT malware families observed in 2017 were *Linux.Lightaidra*, *Trojan.Gen.NPE* and *Linux.Mirai* [41]. For example, the self-propagating multi-stage malware *Mirai* was developed to execute denial-of-service attacks by compromising IoT and infrastructure devices and performing application-level (e.g. HTTP) or network-level flooding over Generic Routing Encapsulation (GRE), Transmission Control Protocol (TCP), or User Datagram Protocol (UDP). *Mirai* works by first installing a loader on the victim's host which creates a multi-threaded server. *Mirai* then scans its environment for open telnet ports to propagate itself to hosts by brute-forcing telnet passwords based on a dictionary. Subsequently, the loader is executed which connects the host to a C2 server to download and execute architecture specific payload. Although *Mirai* does not show persistency mechanisms, the malware provides basic defense functionality such as process termination. Basic obfuscation techniques are performed to cover malicious activity including deletion of downloaded binaries and process name obfuscation. Another example of more sophisticated utilization of obfuscation techniques is ransomware. The wide-spread distribution of malware instances (e.g. WannaCry, Petya, NotPetya, BadRabbit) in 2017 suggests an increased utilization of obfuscation techniques. While the number of new ransomware families has decreased by 63% from 98 families in 2016 to 28 families in 2017, the number of new ransomware variants have increased by 46% from 241.000 to 350.000 indicating a lack of new threat actor groups and less innovation on the one hand but mutation and customization of existing ransomware through obfuscation techniques on the other [41].

1.2 Machine learning to detect unknown threats

For organizations a multitude of technical safeguards is available to detect and prevent cyber threats. Concepts can be broadly categorized into misuse detection and anomaly detection. While misuse detection methods are intended to recognize known attack patterns described by rules, anomaly detection focuses on detecting unusual activity patterns in the observed data [e.g. 17, 18, 42].
The majority of state-of-the-art methods can be classified as misuse detection due to their reliance on rule sets. Rule-based solutions can be further divided into blacklist- and whitelist-based approaches. Blacklist-based methods can be further refined into signature-based and heuristic-based approaches. While signature-based approaches allow to detect threats based on specific threat patterns (e.g. malicious byte sequences), heuristic methods allow for the detection of unknown threats based on an expert-based probabilistic rule sets that describe malicious indicators. Although heuristic approaches often complement signature-based solutions, a major drawback is their susceptibility to high false positive rates. Finally, white-list based approaches usually include policies which allow for the detection of threats based on the deviation from a pre-defined negative baseline configuration (e.g. IP whitelists). An overview of state-of-the art misuse and anomaly-based detection methods is provided by the work of Modi [43] and Mitchell [44].
To overcome challenges with existing approaches (i.e. limited ability to detect unknown threats by signature-based methods and lack of detection accuracy by behavior-based methods), interest has grown in the security community to utilize machine learning as an alternative and more accurate approach to

existing methods for the detection of unknown threats or variants of known threats [e.g. 45–52].
Machine learning provides methods to automatically infer generalized data models based on patterns identified in data. Significant research has been conducted on theory and methods for machine learning over the past decades [e.g. 53–56]. An overview of learning techniques is provided by the work of Duda [57] and Hastie [58] whereas kernel-based learning is more specifically discussed by Mueller [59] and Smola [60].

Generally, machine learning is concerned with solving an optimization problem to find a function $\hat{f}_\theta$ with parameters θ that minimizes the expected risk $R(f)$ (test error):

$$\hat{f} = \underset{f \in \mathcal{H}}{\operatorname{argmin}} R(f) + \Omega(f), \tag{1.1}$$

whereas $f : \mathcal{X} \mapsto \mathcal{R}$ refers to a predictor that learns a real-value mapping of a set of n-dimensional data points $\mathcal{X} = \{x_1, x_2, x_i, ..., x_m \mid x_i \in \mathcal{R}^n\}$ to an output variable $\hat{y} \in \mathcal{R}$. The risk function $R(f) = \int \mathbb{L}(f(x), y) dP(x, y)$ is defined as the expectation of the loss function $\mathbb{L}(f(x), y)$ that penalizes incorrect prediction of $\hat{y} = f(x)$ based on the ground truth mapping y. The most common loss function for classification and anomaly detection is the *squared loss* $\mathbb{L}(f(x), y) = (f(x) - y)^2$ [59, 61].
Unfortunately, the expected risk $R(f)$ cannot be calculated as the underlying joint probability distribution $P(x, y)$ is unknown. However, as an approximation, the empirical risk $R_{emp}(f) = \frac{1}{n} \sum_{i=1}^{n} \mathbb{L}(f(x_i), y_i)$ (training error) can be calculated as the average loss over a sample of training points for a function in function set $\mathcal{H}$. Unfortunately, minimization of the empirical risk is not sufficient to obtain a function which accurately predicts on unseen data instances. Structural risk minimization, originally introduced by Vapnik [62], provides a framework that puts expected risk and empirical risk into a relationship. The framework provides an upper bound for the expected risk $R(f) \leq R_{emp}(f) + C(f, ...)$ using the empirical risk $R_{emp}(f)$ and the capacity term C of the function class of the predictor f. Intuitively, the capacity term reflects model complexity. While predictors with linear decision surface have low capacity, non-linear functions demonstrate high capacity.
The accuracy of a predictor is largely determined by two types of errors: *bias* and *variance*. While the bias error takes into account the deviation of the expected prediction of a model to the true target values, the variance error takes into account the variability of the prediction based on training of a specific function over different training sets. While high capacity function (e.g. non-parametric or non-linear functions) have low bias but high variance (risk of overfitting), low capacity functions (e.g. parametric or linear functions) tend to have high bias but low variance (risk of underfitting). The overall objective is to learn a function with low bias and low variance, i.e. a function with low training error but also generalizes on unseen data. This relationship is commonly known in machine learning as the *bias-variance dilemma*.
Mathematically, the total error $E(x)$ (i.e. expected risk) can be decomposed into bias error, variance error and irreducible error [58, 62]:

$$E(x) = (E[\hat{f}(x)] - f(x))^2 + E[(\hat{f}(x) - E[\hat{f}(x)])^2] + \sigma_e^2 \tag{1.2}$$

As illustrated in Fig. 1.1, a data model that minimizes both types of errors, bias and variance, allows to accurately capture regularities in the training data while generalizing well over unknown data.

To address the bias-variance trade-off the concept of regularization is introduced to obtain a data model that both accurately predicts on training and validation data and also generalizes on unseen test data. To

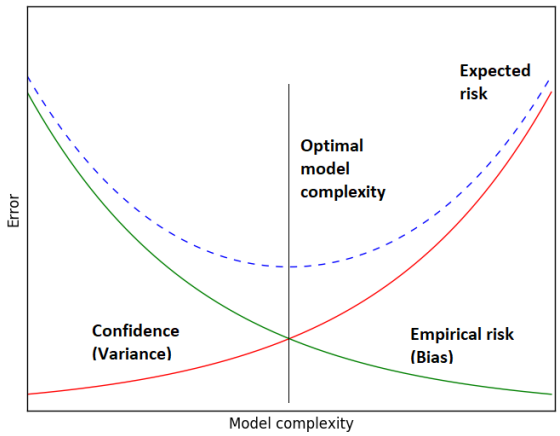


Figure 1.1: Bias-variance trade-off

this end, the minimization problem is extended by a regularization term $\Omega(f) = C \sum_{i=0}^{n} \xi_i$ to penalize function complexity in favor for wrongful prediction for data points x_i through so called slack variables ξ_i which allow misclassification for the benefit of learning function with lower complexity. The parameter C controls the bias-variance trade-off, yielding $C = 0$ for low bias and high variance. An increase of C results in increasing bias and decreasing variance. A suitable C value can be obtained through cross-validation.
Given the overwhelming amount of machine learning methods available, the choice of the right method for the problem at hand depends on its properties. An introduction to various supervised and unsupervised machine learning methods is provided in Chapter 2. Optimization and regularization are two major components for learning accurate data models. Optimization methods can be broadly categorized into iterative and non-iterative methods. In machine learning, gradient-based methods are commonly used for iterative optimization as for example implemented in Multi-layer Perceptron (MLP) networks [63]. Convex optimization is commonly used in non-iterative approaches and is based on *linear* or *quadratic programming*. While iterative methods provide fast convergence, the solution may not converge towards the global minimum. On the other hand, convex optimization allows for convergence to a global minimum but requires objective functions to be convex. Support Vector Machines provide both global optimization as well as regularization [59]. Both properties are described in more detail for the One-class Support Vector Machine (OC-SVM) in Chapter 6. The following paragraphs outline machine learning approaches relevant for the detection of known and unknown threats based on Support Vector Machines for supervised and unsupervised learning.

Supervised learning. In supervised learning a function is trained based on labeled training data whereas each instance of the training data is a tuple consisting of data point x_i represented by its feature space representation $\phi(x_i)$ alongside an output value y_i (e.g. class label). A supervised algorithm constructs a separating hyperplane between two or more classes based on geometric relationships between data points considering their class labels. In this thesis a Support Vector Machine (SVM) is used for classification tasks. In a binary classification scenario, the SVM determines an optimal hyperplane that separates data points from two different classes with maximal margin. Mathematically, this can be

formulated as a quadratic programming optimization problem in its primal form [60]:

$$\begin{aligned} \min_{w,b,\xi} \quad & \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to:} \quad & y_i(\langle w, \phi(x_i)\rangle + b) \leq 1 - \xi_i, \\ & \xi_i \geq 0. \end{aligned} \tag{1.3}$$

Since w lies in the feature space $\mathcal{F}$ (which can have much higher dimensionality than the input space), it cannot be solved directly. By converting the primal form of the problem in 1.3 into its dual form, the optimization problem can be formulated in terms of inner products between data points mapped in the feature space [59].
A binary classification example is depicted in Fig 1.2. In this example, a hyperplane is learned by a two-class SVM using a linear kernel. The margin between data points of two classes is determined by two hyperplanes $H_1 : \langle w, \phi(x_i)\rangle + b \geq 1$ for all $y_i = 1$ and $H_2 : \langle w, \phi(x_i)\rangle + b \leq -1$ for all $y_i = -1$ respectively. An optimal hyperplane represented as the median between the two hyperplanes H_1 and H_2 separates instances from two different classes (i.e. red and blue data points) such that the distance between the hyperplane and the nearest data points from both classes is maximal. Data points on the margin are called support vectors (circled data points) which specify the decision function. The term C in the example denotes a regularization constant which controls the trade-off between margin width and permissible misclassifications in the case of non-separable problems. While large values of C result in smaller margin solutions with less misclassified data points allowed, smaller values of C will increase the margin and allow more misclassifications.

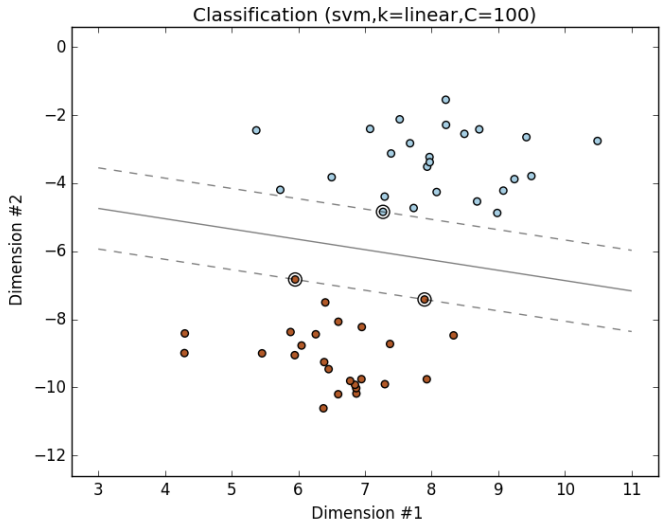


Figure 1.2: Example of supervised learning (Support Vector Machine)

Unsupervised learning. Unlike supervised learning which allows to learn a classifier that separates data instances of two or more classes, unsupervised learning allows to detect outliers from a previously learned model of normality [64]. Decision functions are learned without the use of any label information

based on the assumption that the vast majority of training data is drawn from the same distribution. Outliers are detected as a deviation from a model of normality. In this thesis a OC-SVM [61] is used as a means of novelty detection. The OC-SVM fits a minimal enclosing hypersphere to the data that is characterized by a center θ and a radius R. Mathematically, this can be formulated as a quadratic programming optimization problem:

$$\begin{aligned} \min_{\substack{R\in\mathbb{R} \\ \xi\in\mathbb{R}^n}} \quad & R^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to:} \quad & \|\phi(x_i) - \theta\|^2 \leq R^2 + \xi_i, \\ & \xi_i \geq 0. \end{aligned} \tag{1.4}$$

Given the constraint in Eq.(1.4), minimizing R^2 will minimize the volume of the hypersphere. A novelty detection example is depicted in Fig 1.3. In this example, a hypersphere is learned that separates outliers from the majority population considered normal using a OC-SVM and a Radial Basis Function (RBF) kernel as similarity measure. In the example below, underlying data is bimodal and thus, has two centers of mass. By using a non-linear kernel function (e.g. RBF), the separating hypersphere is fitted to the shape of the underlying distribution without overfitting. The parameter σ (sigma) defines the width of the RBF and determines the influence of support vectors to the decision function. While a small σ results in a sharp decision function, a large σ increases smoothness of the decision function towards a linear decision surface. The parameter ν (nu) in the example below represents a regularization to constant similar to C in Eq. 1.4 to balance the bias-variance trade-off [60]. While a large ν value implies low regularization allowing more samples to be outliers at the benefit of learning a lower complex decision function, a smaller ν value results in strong regularization allowing less samples to be outliers.

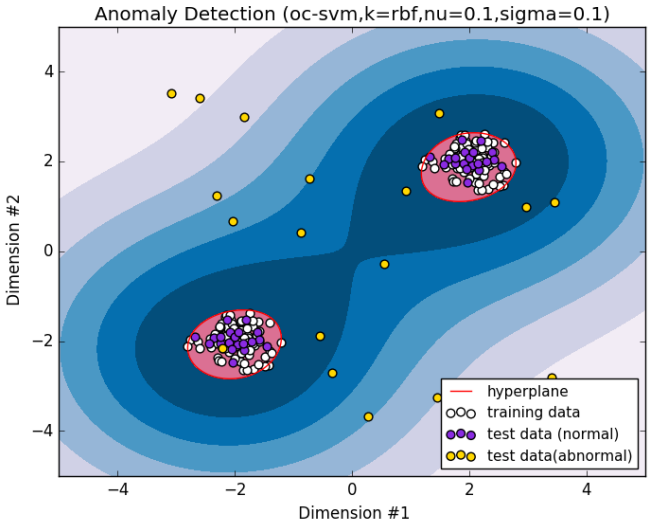


Figure 1.3: Example of unsupervised learning (One-class Support Vector Machine)

Semi-supervised learning. Semi-supervised learning is a class of supervised learning techniques that can make use of unlabeled data for training [65]. The objective is to infer correct labels for unlabeled data points based on the assumption that data points close to each other (continuity assumption) and data points located in the same region in the feature space (cluster assumption) are more likely to share the same labels. Goernitz et al. [51] investigate semi-supervised learning in the context of unknown attack detection and further propose an active learning strategy to select unlabeled candidates for labeling to improve the detection accuracy of the predictor. While semi-supervised learning approaches provide benefits in terms of trade-off between improvement of accuracy by labeling a subset of data, these methods are not included in the scope of this dissertation.

1.3 Data representations for sequential data

Machine learning methods usually operate on vector data. However, in computer and network security, sequential data is oftentimes found. Feature engineering is concerned with the construction of features that describe key characteristics of the underlying data. By using a generic feature map $\phi : \mathcal{X} \mapsto \mathbb{R}^N$ data points can be represented in a N-dimensional vector space $\mathbb{R}^N$ induced by a set of domain-descriptive features.

$$x \mapsto \phi(x) = (\phi_1(x), \phi_2(x), \phi_3(x), ..., \phi_N(x)), \quad 1 \leq N \leq \infty. \tag{1.5}$$

By utilizing data representations, sequential data can be embedded into a feature space in which a decision function is learned based on geometric relationships between data representations of individual sequences. Good data representations comprise non-redundant features which are discriminative to the problem at hand. Oftentimes, the complexity of the data domain causes the volume of the feature space to increase exponentially resulting in sparsely populated regions in the feature space [66]. High dimensionality of input feature vectors exacerbates the learning process, particularly if discriminative information resides in manifolds of the feature space (i.e. if the true function only depends on a small number of features) which may result in high variance of the predictor. Thus, a good data representation allows for efficient storage and processing of features. The following paragraphs provide an overview of data representations that can be used for classification and anomaly detection in the field of computer and network security.

Sequential data representation. A substantial amount of work has been done in developing efficient data structures for the comparison of sequential data such as suffix tries, suffix trees or suffix array [e.g. 67–69]. Suffix tries are space-efficient and allow for comparison of sequences in linear time. However, due to the explicit representation of common path inner nodes in the suffix trie, the practical utilization of this type of data structure is limited to comparably small sequences. A suffix tree is a compressed trie containing all the suffixes of the given text as their keys and positions in the text as their values by merging common paths to a single node. A suffix tree can be constructed in linear time and also allows for linear time comparison of sequences [e.g. 69]. Due to the implicit representation of common paths, suffix tree are more suitable to store and process large sequences - as typically found in network packet payloads. An example of a suffix tree representing the sequence s="ababc$" is depicted in Fig 1.4. This suffix tree data representation can be used to compare sequential similarity between sequences [67]. Fig. 1.4 shows a suffix tree as an example to represent byte sequences.
By traversing a suffix tree, unique features can be efficiently extracted. An intuitive feature type at byte-level are language models. A commonly used type of language model is referred to as k-grams which involves extraction of unique substrings by moving a sliding window of length k over a byte sequence [67].

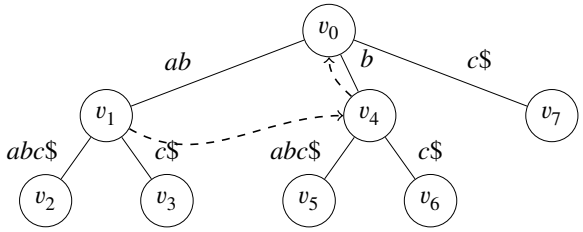


Figure 1.4: Suffix tree to represent byte sequences ("ababc$")

More formally, consider an alphabet to be a finite set of symbols Σ. In network and computer security the alphabet $\Sigma = \{0, ..., 255\}$ usually correspondes to byte values. Let Σ^* denote the set of all possible concatenation of Σ and the set of all possible concatenation of length k to be Σ^k. The k-spectrum of a symbol sequence refers to the set of of all contiguous unique subseqences u of length k generated by an alphabet Σ. Hence, a mapping function ϕ can be define which maps fixed length substrings u contained in string s into a feature space $\mathcal{F}$:

$$\phi : s \longmapsto (\phi_u(s))_{u \in \Sigma^k} \in \mathcal{F}. \tag{1.6}$$

The mapping function $\phi_u(s)$ allows to map a subsequence u contained in s to a binary, count or relative frequency value. A binary mapping function sets the value of a mapped subsequence u to one if u is contained at least once in s. A count mapping function sets the value of the mapped subsequence u to m if u is contained m times in s. Finally, a frequency mapping function sets the value of a mapped subsequence u to $\frac{m}{n-k+1}$ whereas m denotes the number of times u is contained in s and n refers to the length of string s.
It is noteworthy, that by traversing a suffix tree, other types of features can be extracted as well, such as *bag-of-words* or *all-subsequences* features [67].

Syntactic data representation. Network protocol messaging follows a defined context-free grammar. As attacks are commonly delivered at the application-layer, network protocol syntax may carry useful information. By deploying network protocol analysis, syntactic information can be extracted from byte-level sequences. The syntax tree data representation can be used to represent the structure of a sequence generated by a context-free grammars [e.g. 70–72].
A tree T=(V,E) is a directed acyclic graph in which any two vertices $v_1, v_2 \in V$ are connected by at most one path $e_1, ...e_i, ...e_n \; \forall e_i \in E$. A *leaf* or *terminal* node refers to a node without children. An *internal* node is a node with at least one child. A *pre-terminal* node refers to an internal node with only leaf node children attached.
A *rooted tree* is called a tree in which a special labeled node is singled out as the root node. A *labeled tree* is a tree in which each internal node is associated with a label. A *structured tree* is one in which the children of any node are given a fixed ordering. A *syntax tree* represent a specific type of a labeled structured tree.
An *attributed tree* is a labeled, structured tree in which each node v is associated with a finite sequence $s = s_1...s_{|s|}$ of symbols drawn from an alphabet of unique symbols Σ, including the empty sequence. By applying a specified grammar, network protocol analyzers can be used to extract protocol structure from byte sequences with attributes associated to individual protocol elements.
Fig. 1.5 shows an example of a syntax tree for a HTTP request. In this example, the HTTP request is parsed and substrings are associated with network protocol elements.

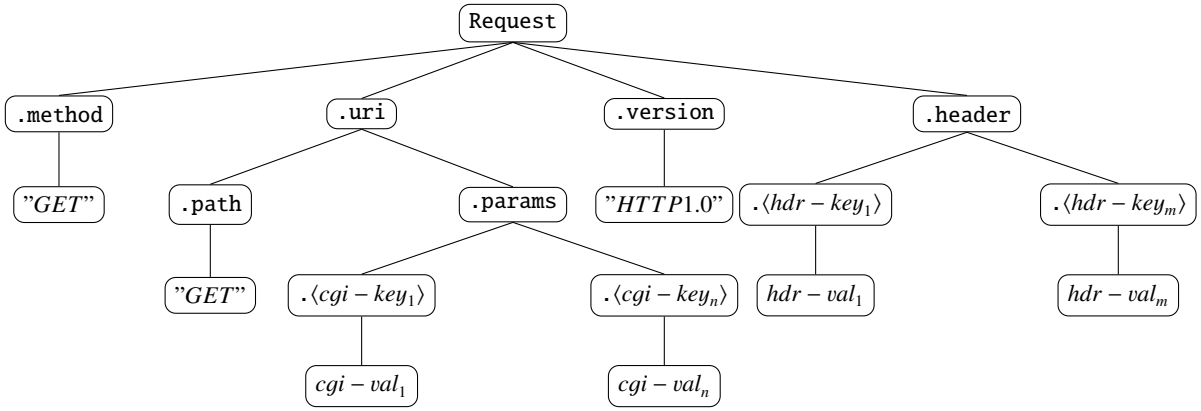


Figure 1.5: Syntax tree to represent syntax structure of byte sequences

The syntax tree data representation can be used to compare structural similarity between sequences [e.g. 71]. In order to compare syntax trees, a corresponding feature map must be defined. A commonly used feature map $\phi(T)$ involves the mapping of all subtrees of a tree T to a feature space $\mathcal{F}$:

$$\phi : T \longmapsto (\phi_S)_{S \in I} \in \mathcal{F}, \tag{1.7}$$

whereas I denotes the set of all possible subtrees. The mapping function returns 1 if a subtree S is a subtree of T and 0 otherwise.

$$\phi_S(T) = \begin{cases} 1 & \text{if S is a subtree of T} \\ 0 & otherwise. \end{cases} \tag{1.8}$$

An alternative feature mapping is based on the idea of k-grams. To this end, consider an alphabet to be a finite set of syntactic tokens Σ. A mapping function ϕ can be defined which maps sequences of syntactic tokens u of length k into a feature space $\mathcal{F}$:

$$\phi : T_s \longmapsto (\phi_u(T_s))_{u \in \Sigma^k} \in \mathcal{F}. \tag{1.9}$$

The mapping function $\phi_u(T_s)$ allows to map a token subsequence u contained in tree T_s (i.e. syntactic representation of sequence s) to a binary, count or relative frequency value. The resulting feature set can be referred to as *token grams*. For $k = 1$, the feature set corresponds to a *bag-of-tokens* data representations. By increasing k the sensitivity for structural differences between two syntax trees increases as well.

Syntax-sequential data representation. The syntax-sequential data representation (c_k-grams) can be considered an extension of the sequential data representation as outlined in Section 1.3. As k-grams can have multiple different protocol contexts in a parsed sequence, the same sequential feature can co-exist in different locations in the geometric space taking into account its protocol context. Fig. 1.6 shows an example of a c_2-gram data representation of an HTTP requests. For example, the bi-gram "er" occurs in the tokens "keyword" (2) as well as in the "Host" (6) context of the HTTP request. Thus, this feature is embedded in two different locations in the feature space.

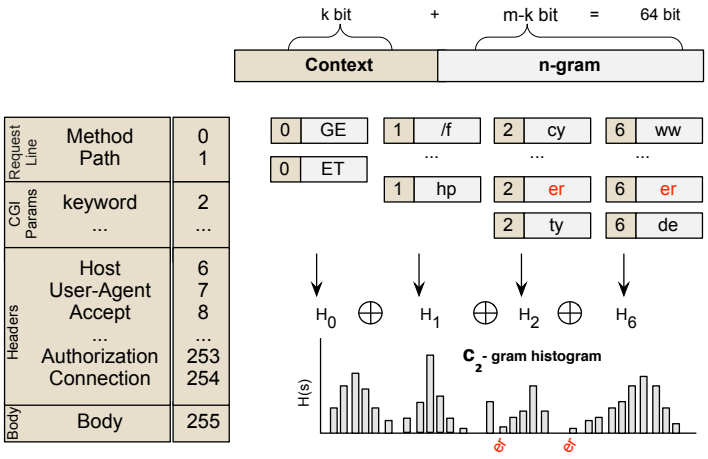


Figure 1.6: Contextual grams (c_2-grams) of an HTTP request

Graph-based data representation. Graphs can be used to represent semantic relationships extracted from sequential data. Graph representations can be used to learn behavior models based on observed user or system activity (i.e. a subject performs specific activities on a specific object).
Let $E = (e_1, ..., e_i, ..., e_m)$ be a list of chronologically ordered events $e_i \in \Sigma$. A session $S = (e_1, ..., e_n) \subseteq E$ can be considered as an ordered set of activities represented by events $e_i \in \Sigma$. An activity model $G = (\Sigma, Q, \hat{q}, T, F)$ can be defined as a directed graph where Σ refers to the universal set of events, Q denotes the set of states of a session (represented as nodes), with $\hat{q}$ being the current state, $T : Q \times \Sigma \rightarrow Q$ being the state transition function used to specify permissible state changes based on the observed events and F being the set of final states. The example in Fig. 1.7 shows a graph which represents a user who performs a series of web requests within a session.

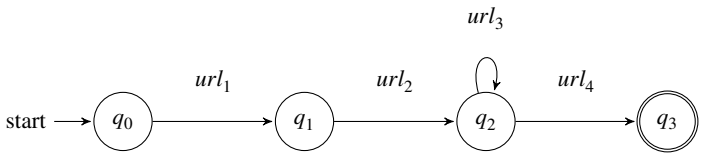


Figure 1.7: Graph representing user activity

In order to compare graphs, a corresponding feature map must be defined. A commonly used feature map $\phi(T)$ involves the mapping of all paths of a graph G to a feature space $\mathcal{F}$:

$$\phi : G \longmapsto (\phi_w)_{w \in W} \in \mathcal{F}, \tag{1.10}$$

whereas W denotes the set of all possible walks (i.e. paths) induced by the set of vertices. The mapping function returns 1 if a path w is contained in G and 0 otherwise.

$$\phi_p(G) = \begin{cases} 1 & \text{if w is a path contained in G} \\ 0 & \textit{otherwise.} \end{cases} \tag{1.11}$$

1.4 Measuring similarity in feature spaces

Machine learning methods are based on data models which represent geometric relationships between data points based on defined similarity measures in a geometric space. Similarity measures can be broadly categorized into *distances*, *coefficients* and *kernels*. Table 1.1 outlines commonly used similarity measures [e.g. 73, 74].

Distance		Coefficient		Kernel	
Name	$d(\mathbf{x}, \mathbf{y})$	Name	$s(\mathbf{x}, \mathbf{y})$	Name	$k(\mathbf{x}, \mathbf{y})$
Minkowski	$(\sum_i \lvert\phi_i(x) - \phi_i(y)\rvert^k)^{\frac{1}{k}}$	Czekanowski	$\frac{2a}{2a+b+c}$	Linear	$\sum_i \phi_i(x)\phi_i(y)$
Mahalanobis	$\sqrt{\sum_i \frac{(\phi_i(x)-\phi_i(y))^2}{\sigma_i^2}}$	Kulczynski	$\frac{1}{2}(\frac{a}{a+b} + \frac{a}{a+c})$	Gaussian	$e^{(\frac{\lVert\phi(\mathbf{x})-\phi(\mathbf{y})\rVert^2}{\sigma^2})}$
Chebychev	$\max_i \lvert\phi_i(x) - \phi_i(y)\rvert$	Jaccard	$\frac{a}{a+b+c}$	Polynomial	$(\sum_i \phi_i(x)\phi_i(y) + c)^d$
Canberra	$\sum_i \frac{\lvert\phi_i(x)-\phi_i(y)\rvert}{\lvert\phi_i(x)\rvert+\lvert\phi_i(y)\rvert}$	Sokal-Sneath	$\frac{a}{a+2(b+c)}$		
Cosine	$\frac{\sum_i \phi_i(x)\phi_i(y)}{\sqrt{\sum_i \phi_i(x)^2}\sqrt{\sum_i \phi_i(y)^2}}$	Sorenson-Dice	$\frac{a}{b+c}$		

Table 1.1: Similarity measures

Distances. A distance $d : X \times X \mapsto \mathbb{R}$ is a a real-valued function over a set of data points X which satisfies the following conditions for any data point $x, y, z \in X$: *non-negativity* (i.e. $d(x, y) \geq 0$), *identity of indiscernibles* (i.e. $d(x, y) = 0 \iff x = y$), *symmetry* (i.e. $d(x, y) = d(y, x)$), and *sub-additivity* (i.e. $d(x, z) \leq d(x, y) + d(y, z)$).
Commonly used distances for approximate string matching include *Hamming* distance [75] and *Levenshtein* distance [76, 77]. However, both distances show limited usability for the application in the network and host security domain. For example, the Hamming distance requires equal length binary strings as input while the Levenshtein distance employs non-linear runtime. By casting sequences into their vector space representations, other distance functions can be used such as *Minkowski*, *Canberra* or *Mahalanobis*.

Similarity coefficients. A similarity coefficient $s : X \times X \mapsto \mathbb{R}$ is a a real-valued function over a set of data points X which reflects similarity between two data points based on four aggregation variables **a**,**b**,**c**, and **d**. Thereby, the variable **a** defines the number of positive matching components in both data points, **b** denotes the number of left mismatches, **c** the number of right mismatches, and **d** represents the number of negative matches (i.e. zero-value variables in both data points). Similarity coefficients do not necessarily satisfy all distance properties. For example, the *Sorenson-Dice* coefficient does not meet the *non-negativity* and *sub-additivity* condition.

Kernels. A kernel $k : X \times X \longmapsto \mathbb{R}$ is a real-valued function over $X \times X$ such that for any two data point $\mathbf{x}, \mathbf{y} \in X$ $k(\mathbf{x}, \mathbf{y}) = \langle\phi(\mathbf{x}), \phi(\mathbf{y})\rangle_{\mathcal{F}}$ for an inner product space $\mathcal{F}$ such that $\forall x \in X\ \phi(x) \in \mathcal{F}$. Thereby, an inner product space $\mathcal{F}$ satisfies *symmetry* ($\langle x, y\rangle = \langle y, z\rangle$), *linearity* ($\langle ax, y\rangle = a\langle x, y\rangle$ and $\langle a + x, y\rangle = \langle a, x\rangle + \langle a, y\rangle$) and *positive semi-definiteness* ($\langle x, x\rangle \geq 0$).
A *Hilbert* space $\mathcal{H}$ refers to a strict inner product space $\mathcal{F}$ in which $\langle x, x\rangle = 0$ iff $x = 0$ and $\mathcal{F}$ is *complete* (i.e. for any $\epsilon > 0$ there exists an N such that for any pair m and n with $n > N$ the $\lvert a_n - a_m\rvert < \epsilon$) and

separable (i.e. for all $h \in \mathcal{F}$ and $\epsilon > 0$ there exists a countable set of elements $h_1, ..., h_i, ...h_n$ of $\mathcal{F}$ such that $|h_i - h| < \epsilon$).
The output of kernel functions can largely differ. For example, a Gaussian kernel is bounded and returns values in the range between 0 and 1. On the other hand, a tree kernel can return much larger values [e.g. 78]. In order to address scale diversity and facilitate learning, kernel normalization can be employed which allows for the projection of kernel values to the same interval. Kernel functions are normalized as defined below:

$$\hat{k}(x, y) = \frac{k(x, y)}{\sqrt{k(x, x) \cdot k(y, y)}}. \tag{1.12}$$

Convolution kernels have been first introduced by Haussler [79] and describe a class of kernel functions that can be used to calculate similarity between structured data objects. Consider a composite data object $\mathbf{x} = (x_1, x_2, ..., x_D)$ consisting of D parts. Let $R(\vec{x}, \mathbf{x}) = R(x_1, x_2, ..., x_D; \mathbf{x})$ be a relation that is true if components $x_1, x_2, ..., x_D$ constitute the data object $\mathbf{x}$. Similarity between two structured data objects $\mathbf{x}$ and $\mathbf{y}$ can be calculated using the R-convolution kernel as introduced by Haussler [79]:

$$k(\mathbf{x}, \mathbf{y}) = [k_1 \times k_2 \times ... \times k_D](\mathbf{x}, \mathbf{y}) = \sum_{\vec{x} \in R(\mathbf{x}),\ \vec{y} \in R(\mathbf{y})} \prod_{d=1}^{D} k_p(x_d, y_d). \tag{1.13}$$

The overall simiarity between two composite data objects can hence be assessed by measuring similarity across all possible decompositions of $\mathbf{x}$ and $\mathbf{y}$. For the remainder of this section, various kernels are outlined that allow to measure similarity between structured data objects which have been shown to be instances of R-convolution kernels [78–80].

String kernel. String kernels allow for the pairwise comparison of sequential data based on common substrings. Based on the set of features extracted using a feature map ϕ, a linear kernel function $k(x, y)$ between two sequences $x, y \in X$ can be defined as follows to measure similarity:

$$k(x, y) = \langle \phi(x), \phi(y) \rangle = \sum_{u \in \mathcal{L}} \phi_u(x) \phi_u(y), \tag{1.14}$$

where $\mathcal{L}$ refers to the underlying language of generated sequences. The language $\mathcal{L} \subseteq \Sigma^n$ includes k-grams - the set of unique substrings of length k generated by the alphabet Σ (c.f. 1.3). The kernel function over k-grams is commonly referred to as *Spectrum* kernel [64, 81]. Complexity of the Spectrum kernel $k(x, y)$ is $O(k|x||y|)$ using dynamic programming. However, using suffix trees the complexity can be reduced to $O(|x| + |y|)$ by pairwise comparison of two sequences x and y through parallel traversal of the corresponding suffix trees $T(x)$ and $T(y)$ along suffix links.
It is noteworthy, that other classes of string kernels exist such as *all-subsequence kernel*, *fixed length-subsequence kernel*, or *gap-weighted subsequence kernel* and *mismatch string kernel* [64, 82, 83]. However, due to their marginal gain compared to an increased runtime complexity, these classes of string kernels are excluded from further consideration.

Tree kernel. A huge amount of data in the network security domain is entangled with specific formats for storage on hosts or exchange of data across a network using network protocols. In network security, for example, data is often transferred using application-level network protocols such as HTTP [26], File Transfer Protocol (FTP) [84], or Simple Mail Transfer Protocol (SMTP) [85] which provides fruitful

information to detect threats.
Tree kernels allow for the pairwise structural comparison of data points based on their tree representations. Two trees T_x and T_y are identical if there exists a bidirectional mapping between their nodes that maintains node labels, parent-child relations as well as ordering of siblings of each internal node. A kernel can be defined by explicitly embedding all finite trees to a vector space $\mathcal{F}$. The *co-rooted subtree kernel* [64, 78] is a commonly used kernel function for structural comparison of trees. Based on the feature map in Eq. 1.7 the following kernel can be defined as follows:

$$\begin{aligned} k(T_x, T_y) &= \langle \phi(T_x), \phi(T_y) \rangle \\ &= \sum_{S \in T} \phi(T_x)\phi(T_y) \\ &= \prod_{i=1}^{d^+ r(T_x)} \phi^r_{S_i}(\tau(ch_i(r(T_x))))\phi^r_{S_i}(\tau(ch_i(r(T_x)))) \\ &= \prod_{i=1}^{d^+ r(T_x)} (k(\tau(ch_i(r(T_x))), \tau(ch_i(r(T_x)))) + 1) \end{aligned} \tag{1.15}$$

whereas T_0 denotes the set of all trees, $\tau(v)$ refers to a subtree of a tree T rooted at node v, $r(T)$ denotes the root of tree T, $ch_i(v)$ denotes the i-th child of a node v and $d^+(v)$ specify the out-degree of node v. The complexity of the calculation is at most $O(min(|T_x|, |T_y|))$.
It is noteworthy, that there are also other, more complex tree kernel functions such as *all-subtree kernel*. However, calculation of these types of kernels is computationally expensive and they are therefore not further considered.

Attributed tree kernel. Structured data is oftentimes annotated with attributes (e.g. Extensible Markup Language (XML) [86]) which may contain information that allow to discriminate between two structurally identical data objects. To this end, this paragraph defines similarity measures that take into account both structural as well as sequential information using *attributed trees*.
Based on a modification of the *co-rooted tree kernel*, an *attributed tree kernel* can be defined to compare similarity between two attributed trees T_x and T_y:

$$\begin{aligned} k(T_x, T_y) &= \langle \phi(T_x), \phi(T_y) \rangle \\ &= \sum_{S \in T} \phi(T_x)\phi(T_y) \\ &= \prod_{i=1}^{d^+ r(T_x)} \phi^r_{S_i}(\tau(ch_i(r(T_x))))\phi^r_{S_i}(\tau(ch_i(r(T_x)))) \\ &= \prod_{i=1}^{d^+ r(T_x)} (k(\tau(ch_i(r(T_x))), \tau(ch_i(r(T_x)))) + \hat{k}(attr(r(T_x)), attr(r(T_y)))), \end{aligned} \tag{1.16}$$

where $\hat{k}(s_1, s_2)$ denotes a normalized sequence kernel function to compare two sequences s_1 and s_2 as introduced in Paragraph 1.3 and $attr(v)$ refers to a function that extracts a sequential attribute associated with an inner node v in the tree. In that context, the kernel function $k(T_x, T_y)$ returns the sequential similarity between attributes of identical labeled structured subtrees. Note, that by using the k-spectrum kernel, the spectrum length should be chosen to meet $k \leq min(\{|attr(v)|\})$. It is clear from the definition above, that similarity calculation over attributed trees increases complexity as attributes of matching

labeled tree nodes have to be compared individually using sequential kernel functions.
However, in the field of network security, complexity of calculation can be reduced as internal nodes of an attributed tree are not associated with attributes. Therefore, calculation of similarity between two parse trees can be limited to pairwise comparison of their pre-terminal nodes associated with corresponding attributes extracted from the network protocol analysis. To this end, the *attributed tree kernel* can be reduced to an *attributed token kernel* [71].
The c_k-gram data representation (c.f. Section 1.3) contains k-grams extracted from parsed attributes annotated by their corresponding protocol token. Thus, the data representation allows for similarity comparison between sequences based on the similarity of substrings with matching syntactical context [72].

Graph kernel. Graph kernels are useful to model behavior (e.g. user or system behavior) or complex data structures, e.g. as typically found in computational chemistry.
Consider a digraph $G = (V, E \subseteq V \times V, v_{start}, v_{end})$ consisting of a set of vertices V and a set of directed edges E whereas each edge $e_i = (v_1, v_2) \in E$ connects two adjacent vertices v_1 and v_2 in G. A digraph is called labeled if each vertex is assigned to a unique label to distinguish it from other vertices. Graph kernels are a specfic type of convolution kernels on pairs of graphs to calculate similarity between graphs which is known to be a difficult problem. It has been shown that calculating a strictly positive definite graph kernel is at least as hard as solving the NP-hard graph isomorphism problem [87].
Alternative approaches include the enumeration of *random walks* by counting the number of common walks in two input graphs G_x and G_y whereas a walk (i.e. path) denotes a sequences of nodes in a graph (with possible repetitions). Conventionally, the walks of length k can be calculated by looking at the k-th power of the adjacency matrix A of each graph as defined below:

$$k(G_x, G_y) = \sum_{i,j=1}^{|V|} \Big[\sum_{k=0}^{\infty} \lambda^k A^k \Big]_{i,j}. \tag{1.17}$$

Drawbacks of this approach are however exponential runtime behavior, halting or tottering due to repetitive node walks. Various alternative kernels have been proposed (e.g. shortest-path kernel, optimal assignment kernel, weighted decomposition kernel, edit-distance kernel, cyclic pattern kernel, graphlet kernel) to calculate graph similarity [64]. However, these approaches are either computationally expensive or do not satisfy the positive semi-definiteness property.
A computational less expensive alternative involves the utilization of the c_k-gram data structure (c.f. Section 1.3) by de-composing the graph into its set of unique fixed-length walks and mapping the graph in the feature space using a feature map as outlined in Eq. 1.10. The similarity between two graphs G_x and G_y can then be determined by calculating the inner product over the set of all possible walks P of length k common in both graphs:

$$\begin{aligned} k(G_x, G_y) &= \langle \phi(G_x), \phi(G_y) \rangle \\ &= \sum_{w \in W} \phi_w(G_x) \phi_w(G_y). \end{aligned} \tag{1.18}$$

However, this approach has one limitation that the calculation of similarity does not account for cycles in a graph as the c_k-gram data structure hashes unique walks only. Therefore, repetitive walks are not considered.

1.5 Thesis outline

This dissertation addresses a major challenge in cyber security research – namely the reliable detection of unknown cyber attacks. The main focal point of this dissertation is the development of machine learning methods to detect unknown attacks in computer networks at the application-layer and furthermore derive useful information to pinpoint unknown vulnerabilities in applications based on the analysis of network packet payloads. To this end, six representative studies are introduced and organized in individual chapters. While the first five contributions focus on network security, the last study demonstrates usability of the proposed methods in other areas of computer and network security such as malware detection. Each of the presented studies addresses a specific research question related to the main focus of this dissertation:

- *To what extent is unsupervised machine learning an alternative to supervised approaches for the detection of network attacks using network packet header features?* – Numerous machine learning methods can be used to detect network attacks. Generally, methods can be distinguished between supervised and unsupervised learning. While supervised learning requires label information to train models, unsupervised learning allows to detect attacks as outliers in terms of deviation from a model or normality. Chapter 2 provides a comparative analysis of various supervised and unsupervised methods on the well-known KDD Cup 1999 dataset [88] over network packet header features to investigate strengths and limitations in terms of capabilities to detect known and unknown attacks in the presence and absence of label information.

- *To what extent does unsupervised machine learning allow for the detection of unknown network attacks in network packet payloads?* – One of the limitations of the KDD Cup 1999 dataset used in the experimental evaluation in Chapter 2 is its focus on network packet header information. The study presented in Chapter 3 extends the work described in Chapter 2 and provides an experimental evaluation of anomaly detection using String kernel measures over language models extracted from both plain-text and binary network packet payloads commonly seen in Supervisory and Data Acquisition (SCADA) network traffic of industrial automation facilities [89].

- *What are the features that maximize accuracy of network attack detection?* – Good features contain discriminative information to learn accurate data models. Oftentimes, the selection of the right set of features is a problem given the extensive amount of features available. For example, while Chapter 2 focuses on network packet header features, experiments in Chapter 3 are based on content byte stream features and experiments in Chapter 5 elaborate on the usefulness of structural features to detect network attacks. In order to investigate which features are most effective for anomaly detection in network security, Chapter 4 introduces a method for automatic feature selection and provides an experimental evaluation to identify features to maximize accuracy of anomaly-based attack detection in network traffic.

- *To what extent does the incorporation of network protocol information in similarity measures improve the detection of application-level attacks?* – Nowadays, the vast majority of network attacks are carried out at the application layer entangled with network protocols. As a continuative study Chapter 5 introduces a SVM-based anomaly detection method that allows for the detection of unknown network attacks based on syntax-sequential features extracted from parsed network packet payloads containing HTTP requests. To this end, this study introduces a novel composite similarity which is used to calculate pairwise similarity of HTTP requests based on the sequential similarity of byte-sequence components of matching syntactic tokens.

- *To what extent does combination of syntax and sequential features improve the detection of application-layer attacks and increase explainability?* – Fast similarity calculation is crucial for the practical use of machine learning in computer and network security. Unfortunately, similarity calculation over composite data structures can be complex due to kernel normalization across components. One way of reducing complexity and increasing speed is to disable normalization. However, this may lead to inaccurate data models due to under-representation of important features or over-representation of unimportant features. Chapter 6 introduces a novel, efficient feature extraction method that allows to store syntactical information along with associated byte-level features and this bridges the gap between network protocol analysis and anomaly detection. The novel data representation can be used by regular kernel measures without additional kernel normalization and furthermore can be used by security analysts to pinpoint unknown vulnerabilities in network protocols and applications.

While the previous five studies focus on network security, the following study is presented in order to demonstrate general applicability of the proposed methods in the field of computer security. Specifically, the contribution addresses the question of how to detect unknown malware instances based on malware behavior. To this end, Chapter 7 investigates the usability of SVM-based classification over features extracted from malware behavior reports to detect novel malware families.

Bibliography

[1] I. S. Consortium. *ISC Internet Domain Survey*. 2017. URL: https://www.isc.org/network/survey (visited on 22/12/2017) (cit. on p. 1).

[2] A. Nordrum. *Popular Internet of Things Forecast of 50 Billion Devices by 2020 Is Outdated*. 2016. URL: https://spectrum.ieee.org/tech-talk/telecom/internet/popular-internet-of-things-forecast-of-50-billion-devices-by-2020-is-outdated (visited on 22/12/2017) (cit. on p. 1).

[3] E. de Argaez. *Internet World Stats*. 2017. URL: http://www.internetworldstats.com (visited on 22/12/2017) (cit. on p. 1).

[4] Statista. *Global number of cyber security incidents from 2009 to 2015 (in millions)*. 2015. URL: https://www.statista.com/statistics/387857/number-cyber-security-incidents-worldwide (visited on 22/12/2017) (cit. on p. 1).

[5] J. Mchugh. "Intrusion and Intrusion Detection". In: *Int. J. Inf. Secur.* 1.1 (Aug. 2001), pp. 14–35. ISSN: 1615-5262 (cit. on pp. 1, 2).

[6] J. Mirkovic et al. *Internet Denial of Service: Attack and Defense Mechanisms (Radia Perlman Computer Networking and Security)*. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2004. ISBN: 0131475738 (cit. on p. 1).

[7] T. Holz. "A Short Visit to the Bot Zoo". In: *IEEE Security and Privacy* 3.3 (May 2005), pp. 76–79. ISSN: 1540-7993 (cit. on p. 1).

[8] NIST. *NIST Information Technology Laboratory - National Vulnerability Database*. 2017. URL: https://nvd.nist.gov (visited on 22/10/2017) (cit. on p. 1).

[9] G. Wurster and P. C. van Oorschot. "The Developer is the Enemy". In: *Proceedings of the 2008 New Security Paradigms Workshop*. NSPW '08. Lake Tahoe, California, USA: ACM, 2008, pp. 89–97. ISBN: 978-1-60558-341-9 (cit. on p. 2).

[10] Symantec. *ISTR - Living off the land and fileless attack techniques*. 2017. URL: https://www.symantec.com/content/dam/symantec/docs/security-center/white-papers/istr-living-off-the-land-and-fileless-attack-techniques-en.pdf (visited on 22/10/2017) (cit. on p. 2).

[11] B. Bashari R., M. Masrom and S. Ibrahim. "Camouflage In Malware: From Encryption To Metamorphism". In: *International Journal of Computer Science And Network Security (IJCSNS)* 12 (2012), pp. 74–83 (cit. on p. 2).

[12] V. Enterprise. *2017 Data Breach Investigations Report*. 2017. URL: http://www.verizonenterprise.com/verizon-insights-lab/dbir/2017 (visited on 22/10/2017) (cit. on p. 2).

[13] D. E. Denning. "An Intrusion-Detection Model". In: *IEEE Transactions on Software Engineering* SE-13.2 (1987), pp. 222–232 (cit. on p. 2).

[14] M. Roesch. “Snort - Lightweight Intrusion Detection for Networks”. In: *Proceedings of the 13th USENIX Conference on System Administration*. LISA ’99. Seattle, Washington: USENIX Association, 1999, pp. 229–238 (cit. on p. 2).

[15] V. Paxson. “Bro: A System for Detecting Network Intruders in Real-time”. In: *Comput. Netw.* 31.23-24 (Dec. 1999), pp. 2435–2463. ISSN: 1389-1286 (cit. on p. 2).

[16] R. Bace and P. Mell. “NIST Special Publication on Intrusion Detection Systems”. In: (2001) (cit. on p. 2).

[17] L. Portnoy, E. Eskin and S. Stolfo. “Intrusion detection with unlabeled data using clustering”. In: *In Proceedings of ACM CSS Workshop on Data Mining Applied to Security (DMSA-2001*. 2001, pp. 5–8 (cit. on pp. 2, 4).

[18] E. Eskin et al. “A Geometric Framework for Unsupervised Anomaly Detection”. In: *Applications of Data Mining in Computer Security*. Ed. by D. Barbará and S. Jajodia. Boston, MA: Springer US, 2002, pp. 77–101 (cit. on pp. 2, 4).

[19] M. V. Mahoney and P. K. Chan. “PHAD: Packet Header Anomaly Detection for Identifying Hostile Network Traffic”. In: 2001 (cit. on p. 2).

[20] S. Hansman and R. Hunt. “A Taxonomy of Network and Computer Attacks”. In: *Comput. Secur.* 24.1 (Feb. 2005), pp. 31–43. ISSN: 0167-4048 (cit. on p. 2).

[21] Rapid7. *The Metasploit Framework*. 2017. URL: https://www.rapid7.com/products/metasploit (visited on 22/10/2017) (cit. on p. 2).

[22] Gallopsled. *CTF Framework and Exploit Development Framework*. 2016. URL: https://github.com/Gallopsled/pwntools (visited on 22/10/2017) (cit. on p. 2).

[23] W. A. Arbaugh, W. L. Fithen and J. McHugh. “Windows of Vulnerability: A Case Study Analysis”. In: *Computer* 33.12 (Dec. 2000), pp. 52–59. ISSN: 0018-9162. DOI: 10.1109/2.889093. URL: http://dx.doi.org/10.1109/2.889093 (cit. on p. 2).

[24] I. X.-F. Research. *CTF Framework and Exploit Development Framework*. 2017. URL: https://securityintelligence.com/apache-struts-2-a-zero-day-quick-draw (visited on 22/10/2017) (cit. on p. 2).

[25] *CVE-2017-5638*. Available from MITRE, CVE-ID CVE-2017-5638. 2017. URL: http://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5638 (visited on 01/04/2018) (cit. on p. 2).

[26] R. Fielding et al. *RFC 2616, Hypertext Transfer Protocol – HTTP/1.1*. 1999. URL: https://tools.ietf.org/html/rfc2616 (visited on 08/04/2018) (cit. on pp. 3, 14).

[27] T. A. S. Foundation. *Apache Struts*. 2018. URL: http://struts.apache.org/ (visited on 05/04/2018) (cit. on p. 3).

[28] Talos. *Content-Type: Malicious - New Apache Struts2 0-day Under Attack*. 2017. URL: http://blog.talosintelligence.com/2017/03/apache-0-day-exploited.html (visited on 05/04/2018) (cit. on p. 3).

[29] M. Christodorescu and S. Jha. “Static Analysis of Executables to Detect Malicious Patterns”. In: *Proceedings of the 12th Conference on USENIX Security Symposium - Volume 12*. SSYM’03. Washington, DC: USENIX Association, 2003, pp. 12–12 (cit. on p. 3).

[30] C. Kruegel, W. Robertson and G. Vigna.
"Detecting Kernel-Level Rootkits Through Binary Analysis".
In: *Proceedings of the 20th Annual Computer Security Applications Conference*. ACSAC '04.
Washington, DC, USA: IEEE Computer Society, 2004, pp. 91–100. ISBN: 0-7695-2252-1
(cit. on p. 3).

[31] E. Kirda et al. "Behavior-based Spyware Detection".
In: *Proceedings of the 15th Conference on USENIX Security Symposium - Volume 15*.
USENIX-SS'06. Vancouver, B.C., Canada: USENIX Association, 2006 (cit. on p. 3).

[32] M. Sharif et al. "Eureka: A Framework for Enabling Static Malware Analysis".
In: *Computer Security - ESORICS 2008*. Ed. by S. Jajodia and J. Lopez.
Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 481–500 (cit. on p. 3).

[33] U. Bayer et al. "Dynamic Analysis of Malicious Code".
In: *Journal in Computer Virology* 2.1 (2006), pp. 67–77 (cit. on p. 3).

[34] U. Bayer et al. "Dynamic Analysis of Malicious Code".
In: *Journal in Computer Virology* 2.1 (2006), pp. 67–77 (cit. on p. 3).

[35] C. Willems, T. Holz and F. Freiling.
"Toward Automated Dynamic Malware Analysis Using CWSandbox".
In: *IEEE Security and Privacy* 5.2 (Mar. 2007), pp. 32–39. ISSN: 1540-7993 (cit. on pp. 3, 97).

[36] B. Anderson et al. "Graph-based Malware Detection Using Dynamic Analysis".
In: *J. Comput. Virol.* 7.4 (Nov. 2011), pp. 247–258. ISSN: 1772-9890 (cit. on p. 3).

[37] M. Egele et al. "A Survey on Automated Dynamic Malware-analysis Techniques and Tools".
In: *ACM Comput. Surv.* 44.2 (Mar. 2008), 6:1–6:42. ISSN: 0360-0300 (cit. on p. 3).

[38] AV-Test. *Security Report 2016/17*. 2017.
URL: https://www.av-test.org/fileadmin/pdf/security_report/AV-TEST_Security_Report_2016-2017.pdf (visited on 22/10/2017) (cit. on p. 3).

[39] T. Holz et al.
"Measurements and Mitigation of Peer-to-peer-based Botnets: A Case Study on Storm Worm".
In: *Proceedings of the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats*.
LEET'08. San Francisco, California: USENIX Association, 2008, 9:1–9:9 (cit. on p. 3).

[40] I. You and K. Yim. "Malware Obfuscation Techniques: A Brief Survey". In: *2010 International Conference on Broadband, Wireless Computing, Communication and Applications*. 2010,
pp. 297–300 (cit. on p. 4).

[41] Symantec. *Internet Security Threat Report 2018*. 2018.
URL: https://www.symantec.com/content/dam/symantec/docs/reports/istr-23-2018-en.pdf (visited on 05/04/2017) (cit. on p. 4).

[42] K. Leung and C. Leckie.
"Unsupervised Anomaly Detection in Network Intrusion Detection Using Clusters".
In: *Proceedings of the Twenty-eighth Australasian Conference on Computer Science - Volume 38*.
ACSC '05. Newcastle, Australia: Australian Computer Society, Inc., 2005, pp. 333–342.
ISBN: 1-920-68220-1 (cit. on p. 4).

[43] C. Modi et al. "A survey of intrusion detection techniques in Cloud".
In: *Journal of Network and Computer Applications* 36.1 (2013), pp. 42 –57 (cit. on p. 4).

[44] R. Mitchell and I.-R. Chen.
"A Survey of Intrusion Detection Techniques for Cyber-physical Systems".
In: *ACM Comput. Surv.* 46.4 (Mar. 2014), 55:1–55:29 (cit. on p. 4).

[45] C. Kruegel and G. Vigna. "Anomaly Detection of Web-based Attacks".
In: *Proceedings of the 10th ACM Conference on Computer and Communications Security*.
CCS '03. Washington D.C., USA: ACM, 2003, pp. 251–261. ISBN: 1-58113-738-9.
DOI: 10.1145/948109.948144. URL: http://doi.acm.org/10.1145/948109.948144
(cit. on pp. 5, 59).

[46] K. Wang and S. J. Stolfo. "Anomalous Payload-Based Network Intrusion Detection". In: *RAID*.
2004 (cit. on pp. 5, 37, 77).

[47] K. Wang, J. J. Parekh and S. J. Stolfo.
"Anagram: A Content Anomaly Detector Resistant to Mimicry Attack".
In: *Proceedings of the 9th International Conference on Recent Advances in Intrusion Detection*.
RAID'06. Hamburg, Germany: Springer-Verlag, 2006, pp. 226–248 (cit. on pp. 5, 77).

[48] P. Laskov et al. *Learning Intrusion Detection: Supervised or Unsupervised?*
Ed. by F. Roli and S. Vitulano. Berlin, Heidelberg: Springer Berlin Heidelberg, 2005, pp. 50–57.
ISBN: 978-3-540-31866-8. DOI: 10.1007/11553595_6.
URL: https://doi.org/10.1007/11553595_6 (cit. on p. 5).

[49] K. Rieck et al. "Learning and Classification of Malware Behavior".
In: DIMVA '08 (2008), pp. 108–125 (cit. on p. 5).

[50] R. Sommer and V. Paxson.
"Outside the Closed World: On Using Machine Learning for Network Intrusion Detection".
In: *2010 IEEE Symposium on Security and Privacy*. 2010, pp. 305–316 (cit. on p. 5).

[51] N. Görnitz et al. "Toward Supervised Anomaly Detection".
In: *J. Artif. Int. Res.* 46.1 (Jan. 2013), pp. 235–262. ISSN: 1076-9757 (cit. on pp. 5, 9).

[52] A. L. Buczak and E. Guven. "A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection".
In: *IEEE Communications Surveys Tutorials* 18.2 (2016), pp. 1153–1176 (cit. on p. 5).

[53] R. A. Fisher. "The Use of Multiple Measurements in Taxonomic Problems".
In: *Annals of Eugenics* 7.7 (1936), pp. 179–188 (cit. on p. 5).

[54] M. Rosenblatt. "Remarks on Some Nonparametric Estimates of a Density Function".
In: *The Annals of Mathematical Statistics* 27.3 (1956), pp. 832–837 (cit. on p. 5).

[55] V. N. Vapnik and A. Y. Chervonenkis.
"On the Uniform Convergence of Relative Frequencies of Events to Their Probabilities".
In: *Theory of Probability and its Applications* 16.2 (1971), pp. 264–280 (cit. on p. 5).

[56] Y. Lecun et al.
"Learning Algorithms For Classification: A Comparison On Handwritten Digit Recognition".
In: *Neural Networks: The Statistical Mechanics Perspective*. World Scientific, 1995, pp. 261–276
(cit. on p. 5).

[57] R. O. Duda, P. E. Hart and D. G. Stork. *Pattern Classification (2Nd Edition)*.
Wiley-Interscience, 2000 (cit. on p. 5).

[58] T. Hastie, R. Tibshirani and J. Friedman. *The Elements of Statistical Learning*.
Springer Series in Statistics. New York, NY, USA: Springer New York Inc., 2001 (cit. on p. 5).

[59] K.-R. Mueller et al. “An Introduction to Kernel-Based learning Algorithms”. In: *IEEE Transactions on Neural Networks* 12.2 (2001), pp. 181–202 (cit. on pp. 5–7).

[60] B. Schoelkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. Cambridge, MA, USA: MIT Press, 2001 (cit. on pp. 5, 7, 8).

[61] D. Tax and R. Duin. “Data domain description by Support Vectors”. In: *Proc. ESANN*. Ed. by M. Verleysen. Brussels: D. Facto Press, 1999, pp. 251–256 (cit. on pp. 5, 8).

[62] V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag New York, Inc., 1995 (cit. on p. 5).

[63] Y. Lecun et al. “Gradient-based learning applied to document recognition”. In: *Proceedings of the IEEE* 86.11 (1998), pp. 2278–2324 (cit. on p. 6).

[64] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. New York, NY, USA: Cambridge University Press, 2004. ISBN: 0521813972 (cit. on pp. 7, 14–16).

[65] O. Chapelle, B. Schlkopf and A. Zien. *Semi-Supervised Learning*. 1st. The MIT Press, 2010 (cit. on p. 9).

[66] R. E. Bellman. *Adaptive Control Processes: A Guided Tour*. Ed. by R. E. Bellman. MIT Press, 1961. ISBN: 9780691079011 (cit. on p. 9).

[67] K. Rieck and P. Laskov. “Language models for detection of unknown attacks in network traffic”. In: *Journal in Computer Virology* 2.4 (2007), pp. 243–256 (cit. on pp. 9, 10).

[68] K. Rieck et al. *Method and apparatus for automatic comparison of data sequences using local and global relationships*. US Patent 8,271,403. 2012 (cit. on p. 9).

[69] E. Ukkonen. “On-line Construction of Suffix Trees”. In: *Algorithmica* 14.3 (Sept. 1995), pp. 249–260. ISSN: 0178-4617 (cit. on p. 9).

[70] C. Bockermann, M. Apel and M. Meier. “Learning SQL for Database Intrusion Detection Using Context-Sensitive Modelling (Extended Abstract)”. In: *Proceedings of the 6th International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. DIMVA ’09. Como, Italy: Springer-Verlag, 2009, pp. 196–205. ISBN: 978-3-642-02917-2 (cit. on p. 10).

[71] P. Duessel et al. “Incorporation of Application Layer Protocol Syntax into Anomaly Detection”. In: *Information Systems Security*. Ed. by R. Sekar and A. K. Pujari. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, pp. 188–202 (cit. on pp. 10, 11, 16, 77).

[72] P. Duessel et al. “Detecting Zero-day Attacks Using Context-aware Anomaly Detection at the Application-layer”. In: *Int. J. Inf. Secur.* 16.5 (Oct. 2017), pp. 475–490. ISSN: 1615-5262 (cit. on pp. 10, 16).

[73] K. Rieck and P. Laskov. “Linear-Time Computation of Similarity Measures for Sequential Data”. In: *J. Mach. Learn. Res.* 9 (June 2008), pp. 23–48. ISSN: 1532-4435 (cit. on p. 13).

[74] S.-S. Choi, S.-H. Cha and C. C. Tappert. “A survey of binary similarity and distance measures”. In: *Journal of Systemics, Cybernetics and Informatics* 8.1 (2010), pp. 43–48 (cit. on p. 13).

[75] R. W. Hamming. “Error detecting and error correcting codes”. In: *The Bell System Technical Journal* 29.2 (1950), pp. 147–160 (cit. on p. 13).

[76] V. Levenshtein. “Binary Codes Capable of Correcting Deletions, Insertions and Reversals”. In: *Soviet Physics Doklady* 10 (1966), p. 707 (cit. on p. 13).

[77] L. Yujian and L. Bo. “A Normalized Levenshtein Distance Metric”. In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 29.6 (June 2007), pp. 1091–1095 (cit. on p. 13).

[78] M. Collins and N. Duffy. “Convolution Kernels for Natural Language”. In: *Advances in Neural Information Processing Systems 14 — Proceedings of the 2001 Neural Information Processing Systems Conference (NIPS 2001), December 3-8, 2001, Vancouver, British Columbia, Canada*. Ed. by T. G. Dietterich, S. Becker and Z. Ghahramani. 2002, pp. 625–632 (cit. on pp. 14, 15).

[79] D. Haussler. *Convolution Kernels on Discrete Structures*. Technical Report UCS-CRL-99-10. University of California at Santa Cruz, 1999. URL: http://citeseer.ist.psu.edu/haussler99convolution.html (cit. on p. 14).

[80] S. V. N. Vishwanathan et al. “Graph Kernels”. In: *J. Mach. Learn. Res.* 11 (Aug. 2010), pp. 1201–1242. ISSN: 1532-4435. URL: http://dl.acm.org/citation.cfm?id=1756006.1859891 (cit. on p. 14).

[81] C. Leslie, E. Eskin and W. S. Noble. “The spectrum kernel: A string kernel for SVM protein classification”. In: *Proceedings of the Pacific Symposium on Biocomputing*. Vol. 7. 2002, pp. 566–575 (cit. on p. 14).

[82] H. Lodhi et al. “Text Classification Using String Kernels”. In: *Journal of Machine Learning Research* 2 (Mar. 2002), pp. 419–444 (cit. on p. 14).

[83] C. Leslie et al. “Mismatch String Kernels for SVM Protein Classification”. In: *Proceedings of the 15th International Conference on Neural Information Processing Systems*. NIPS’02. Cambridge, MA, USA: MIT Press, 2002, pp. 1441–1448 (cit. on p. 14).

[84] J. Postel and J. Reynolds. *RFC 959, FILE TRANSFER PROTOCOL (FTP)*. 1985. URL: https://tools.ietf.org/html/rfc959 (visited on 08/04/2018) (cit. on p. 14).

[85] J. Postel. *RFC 821, SIMPLE MAIL TRANSFER PROTOCOL*. 1982. URL: https://tools.ietf.org/html/rfc821 (visited on 08/04/2018) (cit. on p. 14).

[86] W3C. *Extensible Markup Language (XML) 1.0 (Fifth Edition)*. 2008. URL: http://www.w3.org/TR/xml (visited on 08/04/2018) (cit. on p. 15).

[87] T. Gärtner, P. Flach and S. Wrobel. “On Graph Kernels: Hardness Results and Efficient Alternatives”. In: *Learning Theory and Kernel Machines: 16th Annual Conference on Learning Theory and 7th Kernel Workshop, COLT/Kernel 2003, Washington, DC, USA, August 24-27, 2003. Proceedings*. Ed. by B. Schölkopf and M. K. Warmuth. Springer Berlin Heidelberg, 2003, pp. 129–143 (cit. on p. 16).

[88] M. L. Labs. *1998 DARPA Intrusion Detection Evaluation*. 1998. URL: http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/index.html (visited on 22/10/2017) (cit. on pp. 17, 27).

[89] B. Galloway and G. P. Hancke. “Introduction to Industrial Control Networks”. In: *IEEE Communications Surveys Tutorials* 15.2 (2013), pp. 860–880 (cit. on p. 17).

[90] F. Fleuret and I. Guyon. “Fast Binary Feature Selection with Conditional Mutual Information”. In: *Journal of Machine Learning Research* 5 (2004), pp. 1531–1555 (cit. on p. 51).

[91] K. Wang, G. Cretu and S. J. Stolfo.
"Anomalous Payload-based Worm Detection and Signature Generation".
In: *Proceedings of the 8th International Conference on Recent Advances in Intrusion Detection*. RAID'05. Seattle, WA: Springer-Verlag, 2006, pp. 227–246 (cit. on p. 77).

[92] P. Baecher et al. "The Nepenthes Platform: An Efficient Approach to Collect Malware". Englisch.
In: *Recent advances in intrusion detection : 9th International Symposium, RAID 2006, Hamburg, Germany, September 20 - 22, 2006; proceedings*. Vol. 4219. Springer, 2006, pp. 165–184
(cit. on p. 97).

[93] Avira. *AntiVir PersonalEdition Classic*. 2007.
URL: http://www.avira.de/en/products/personal.html (visited on 22/10/2017)
(cit. on p. 97).

CHAPTER 2

Learning intrusion detection: supervised or unsupervised?

2.1 Introduction

Machine learning techniques have been gaining increasing attention in the intrusion detection community to address well-known limitations of state-of-the-art misuse-based attack detection methods. Main goal of this work is to investigate the trade-offs between different machine learning techniques in their application to network intrusion detection as an alternative to misuse-based attack detection. Motivated by the fact that label information is difficult to obtain, particularly for novel attack types, this contribution provides an experimental evaluation of supervised and unsupervised techniques to compare detection performance and investigates to what extent label information is required to obtain higher detection accuracy. In order to fairly compare methods, this contribution casts experiments into a unified experimental setup that allows to apply model selection for both supervised and unsupervised learning. Experiments are carried out on the well-known KDD Cup 1999 data set [88] in which data instances are mostly characterized by network packet header features at the Transmission Control Protocol/ Internet Protocol (TCP/IP) layer. Evaluation of the algorithms is carried out under two different scenarios. Under the first scenario, training data and test data are drawn from the same distribution (i.e. training and test samples share attack instances) to baseline detection of known attacks. Under the second scenario, attack data is drawn from a distribution that is different from the training data to evaluate the capabilities of the methods to detect unknown attacks.

2.2 Publication

Learning intrusion detection: supervised or unsupervised?

Pavel Laskov, Patrick Düssel, Christin Schäfer and Konrad Rieck

Fraunhofer-FIRST.IDA,
Kekuléstr. 7,
12489 Berlin, Germany
{laskov,duessel,christin,rieck}@first.fhg.de

Abstract. Application and development of specialized machine learning techniques is gaining increasing attention in the intrusion detection community. A variety of learning techniques proposed for different intrusion detection problems can be roughly classified into two broad categories: supervised (classification) and unsupervised (anomaly detection and clustering). In this contribution we develop an experimental framework for comparative analysis of both kinds of learning techniques. In our framework we cast unsupervised techniques into a special case of classification, for which training and model selection can be performed by means of ROC analysis. We then investigate both kinds of learning techniques with respect to their detection accuracy and ability to detect unknown attacks.

1 Introduction

Intrusion detection techniques are usually classified into misuse detection and anomaly detection [1]. Anomaly detection focuses on detecting unusual activity patterns in the observed data [2–6]. Misuse detection methods are intended to recognize known attack patterns. Signature-based misuse detection techniques are currently most widely used in practice; however, interest is growing in the intrusion detection community to application of advances machine learning techniques [7–10]. Not uncommon is also a combination of anomaly and misuse detection in a single intrusion detection system.

To decide which learning technique(s) is to be applied for a particular intrusion detection system, it is important to understand the role the label information plays in such applications. The following observations should be considered:

1. Labels can be extremely difficult or impossible to obtain. Analysis of network traffic or audit logs is very time-consuming and usually only a small portion of the available data can be labeled. Furthermore, in certain cases, for example at a packet level, it may be impossible to unambiguously assign a label to a data instance.
2. In a real application, one can never be sure that a set of available labeled examples covers all possible attacks. If a new attack appears, examples of it may not have been seen in training data.

The main goal of this work is to investigate the tradeoffs between supervised and unsupervised techniques in their application to intrusion detection systems. To this end, we develop an experimental setup in which such techniques can be fairly compared. Our setup is based on the well-known KDD Cup 1999 data set [11]. Since a typical application of a supervised learning method involves model selection, we have built in the same step into unsupervised methods. Performance of both groups of methods is evaluated based on the analysis of the receiver operator characteristic (ROC) curve. The details of our experimental setup are presented in Sec. 2.

Evaluation of several representative supervised and unsupervised learning algorithms, briefly reviewed in Sec. 3, is carried out under the following two scenarios. Under the first scenario, an assumption that training and test data come from the same (unknown) distribution is fulfilled. Under the second scenario, we violate this assumption by taking a data set in which attacks unseen in training data are present in test data. This is a typical scheme to test the ability of an IDS to cope with unknown attacks. The experimental results are presented in Sec. 4.

2 Experimental Setup

2.1 Data source

The KDD Cup 1999 data set [11] is a common benchmark for evaluation of intrusion detection techniques. It comprises a fixed set of connection-based features. The majority of instances in this set (94%, 4898430 instances) has been extracted from the DARPA 1998 IDS evaluation [12]. The remaining fraction of data (6%, 311029 instances) was additionally extracted from the extended DARPA 1999 IDS evaluation [13]. A detailed description of the available features and attack instances can be found in [14, 6].

2.2 Preprocessing

The KDD Cup data set suffers from two major flaws in distribution of data which can bias comparative experiments:

1. The attack rate within the KDD Cup data set is unnatural. About 80% of all instances correspond to attacks, since all one-packet attacks, e.g. the `smurf` attack, are treated as full-value connections and are represented as individual instances.
2. The attack distribution within the KDD Cup data set is highly unbalanced. It is dominated by probes and denial-of-service attacks, which cover millions of instances. The most interesting and dangerous attacks, e.g. the `phf` or `imap` attacks, are grossly under-represented.

In order to cope with these artifacts we preprocess KDD Cup data in order to achieve (a) a fixed attack rate and (b) a balanced distribution of attack and service types.

At the first level of preprocessing, the attack data is split into disjoint partitions containing only one attack type. The normal data is split into disjoint partitions containing only one service type. These partitions are merged into the three disjoint sets of equal length: the training data D_{train}, the validation data D_{val} and the test data D_{test}. This procedure ensures the presence of each attack and service type in the three data partitions.

At the second level of preprocessing samples of 2000 instances are randomly drawn from the training, validation and testing data sets. The sampling procedure enforces a fixed attack rate of 5% and attempts to preserve balanced attack and service type distributions.

The data with "known attacks" is generated from the DARPA 1998 part of the KDD Cup data set. The data with "unknown attacks" has the test part sampled from the DARPA 1999 part of the KDD Cup data set. The attacks in both data sets are listed in Table 1.

2.3 Metric Embedding

The set of features present in the KDD Cup data set contains categorical and numerical features of different sources and scales. An essential step for handling such data is *metric embedding* which transforms the data into a metric space. Our embedding is a two-stage procedure similar to [3, 2].

Embedding of categorical features. Each categorical feature expressing m possible categorical values is transformed to a value in $\mathbb{R}^m$ using a function e that maps the j-th value of the feature to the j-th component of an m-dimensional vector:

$$e(x_i) = \underbrace{(0, \ldots, 1, \ldots, 0)}_{1 \text{ at Position } j} \quad \text{if } x_i \text{ equals value } j$$

Scaling of features. Both the numerical and the embedded categorical features are scaled with respect to each feature's mean μ and standard deviation σ:

$$n(x_i) = \frac{x_i - \mu}{\sigma}$$

"Known" Attack Types	"Unknown" Attack Types
back buffer_overflow ftp_write guess_passwd imap ipsweep land loadmodule multihop neptune nmap perl phf pod portsweep rootkit satan smurf spy teardrop warezclient warezmaster	apache2 httptunnel mailbomb mscan named processtable ps saint sendmail snmpgetattack snmpguess sqlattack updstorm worm xlock xsnoop xterm

Table 1. Distribution of attack types in the experiments.

2.4 Model Selection

Model selection is performed by training a supervised algorithm on a training set D_{train} and evaluating the accuracy on 10 validation sets D_{val} generated as described in Sec. 2. For unsupervised algorithms only evaluation is performed. The criterion for evaluating the accuracy is the area under the ROC curve, computed for the false-positive interval [0, 0.1].

3 Methods

In the following we briefly describe the algorithms used in our experiments.

3.1 Supervised Algorithms

C4.5. The C4.5 algorithm [**?**] performs inference of decision trees using a set of conditions over the attributes. Classification of new examples is carried out by applying the inferred rules. Although the original algorithms contains numerous free parameters, only the number of bootstrap iterations was used in our evaluation.

***k*-Nearest Neighbor.** The k-Nearest Neighbor is a classical algorithm (e.g. [19]) that finds k examples in training data that are closest to the test example and assigns the most frequent label among these examples to the new example. The only free parameter is the size k of the neighborhood.

Multi-Layer Perceptron. Training of a multi-layer perceptron involves optimizing the weights for the activation function of neurons organized in a network architecture. The global objective function is minimized using the RPROP algorithm (e.g. [16]). The free parameter is the number of hidden neurons.

Regularized discriminant analysis. Assuming both classes of examples are normally distributed, a Bayes-optimal separating surface is a hyperplane (LDA), if covariance matrices are the same, or a quadratic surface otherwise (QDA). A gradual morph between the two cases can be implemented by using a regularization parameter γ [17]. Another free parameter λ controls the addition of identity matrix to covariance matrices.

Fisher Linear Discriminant. Fisher Linear Discriminant constructs a separating hyperplane using a direction that maximizes inter-class variance and minimized the intra-class variance for the projection of the training points on this direction (e.g. [19]). The free parameter is the tradeoff between the norm of the direction and the "strictness" of projection.

Linear Programming Machine and Support Vector Machine. Linear Programming Machine (LPM) and Support Vector Machine (SVM) construct a hyperplane of the minimal norm which separates the two classes of training examples (e.g. [**?**]). LPM uses the 1-norm, SVM uses the 2-norm. Furthermore, SVM apply a non-linear mapping to construct a hyperplane in a feature space. In our experiments, radial basis functions are used, their complexity controlled by the width parameter w. Another parameter C controls the tradeoff between the norm of a hyperplane and the separation accuracy.

3.2 Unsupervised Algorithms

γ-Algorithm. The γ-algorithm is a recently proposed graph-based outlier detection algorithm [18]. It assigns to every example the γ-score which is the mean distance to the example's k nearest neighbors. The free parameter is k.

***k*-Means Clustering.** k-Means clustering is a classical clustering algorithm (e.g. [19]). After an initial random assignment of example to k clusters, the centers of clusters are computed and the examples are assigned to the clusters with the closest centers. The process is repeated until the cluster centers do not significantly change. Once the cluster assignment is fixed, the mean distance of an example to cluster centers is used as the score. The free parameter is k.

Single Linkage Clustering. Single linkage clustering [2] is similar to k-Means clustering except that the number of clusters is controlled by the distance parameter W: if the distance from an example to the nearest cluster center exceeds W a new cluster is set.

Quarter-sphere Support Vector Machine. The quarter-sphere SVM [5, 6] is an anomaly detection method based on the idea of fitting a sphere onto the center of mass of data. An anomaly score is defined by the distance of a data point from the center of the sphere. Choosing a threshold for the attack scores determines the radius of the sphere enclosing normal data points.

4 Results

The supervised and the unsupervised algorithms are evaluated separately on the data with known and unknown attacks. The results are shown in Figs. 1 and 2 respectively. The ROC curves are averaged over 30 runs of each algorithm by fixing a set of false-positive rate values of interest and computing the means and the standard deviations of true-positive rate values over all runs for the values of interest.

The supervised algorithms in general exhibit excellent classification accuracy on the data with known attacks. The best results have been achieved by the C4.5 algorithm which attains the 95% true positive rate at 1% false-positive

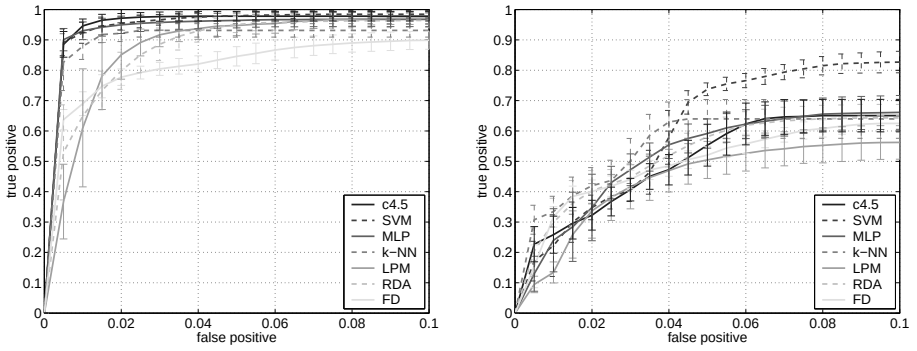


Fig. 1. ROC-curves obtained with the supervised methods on the data sets with known (left) and unknown (right) attacks.

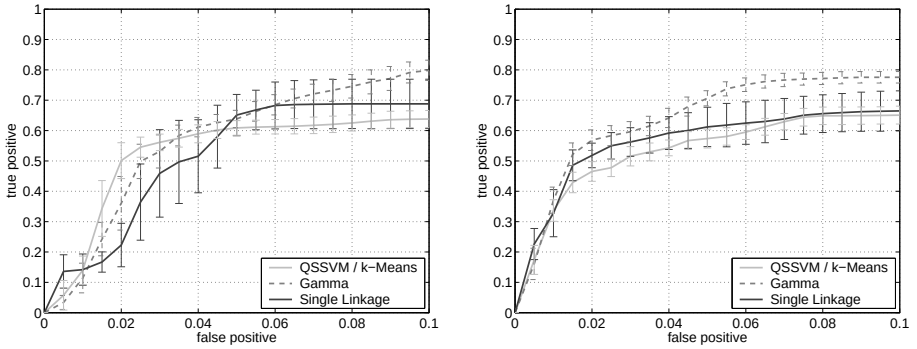


Fig. 2. ROC-curves obtained with the unsupervised methods on the data sets with known (left) and the unknown (right) attacks.

rate. The next two best algorithms are the MLP and the SVM, both non-linear, followed by the local k-Nearest Neighbor algorithm. The difference between the four best methods is marginal. The worst results were observed with the three linear algorithms. One can thus conclude that a decision boundary between the attack and the normal data in KDD Cup features is non-linear, and is best learned by non-linear algorithms or their approximations.

The accuracy of supervised algorithms deteriorates significantly if unknown attacks are present in the test data, as can be seen in the right part of Fig. 1. Not all algorithms generalize equally well to the data with unknown attacks. The best results (with a significant margin) are attained by the SVM, which can be attributed to the fact that the free parameters of this algorithm are motivated by learning-theoretic arguments aimed at maintaining an ability to generalize to unseen data. The next best contestant is the k-Nearest Neighbor algorithm

which possesses the most similarity to the unsupervised methods. The remaining algorithm perform approximately equally.

The unsupervised algorithms exhibit no significant difference in performance between known and unknown attacks. This result is not unexpected: in fact, in all data sets the attacks are unknown to the algorithms – the two data sets differ merely in the set of attacks contained in them. Among the algorithms the preference should be given to the γ-algorithm which performs especially well on the "unknown" data set. The accuracy of unsupervised algorithms on both data sets is approximately the same as that of supervised algorithms on the "unknown" data set.

5 Conclusions

We have presented an experimental framework in which supervised and unsupervised learning methods can be evaluated in an intrusion detection application. Our experiments demonstrate that the supervised learning methods significantly outperform the unsupervised ones if the test data contains no unknown attacks. Furthermore, among the supervised methods, the best performance is achieved by the non-linear methods, such as SVM, multi-layer perceptrons, and the rule-based methods. In the presence of unknown attacks in the test data, the performance of all supervised methods drops significantly, SVM being the most robust to the presence of unknown attacks.

The performance of unsupervised learning is not affected by unknown attacks and is similar to the performance of the supervised learning under this scenario. This makes the unsupervised methods, which do not require a laborious labelling process, a clear forerunner for practical purposes if unknown attacks can be expected.

Our findings suggest that the problem of test data being drawn from a different distribution cannot be solved within the purely supervised or unsupervised techniques. An emerging field of semi-supervised learning offers a promising direction of future research.

Acknowledgements. The authors gratefully acknowledge the funding from *Bundesministerium für Bildung und Forschung* under the project MIND (FKZ 01-SC40A), We would like to thank Stefan Harmeling and Klaus-Robert Müller for fruitful discussions and valuable help in the implementation of the algorithms.

References

1. Bace, R., Mell, P.: NIST special publication on intrusion detection systems. National Institute of Standards and Technology (2001)
2. Portnoy, L., Eskin, E., Stolfo, S.: Intrusion detection with unlabeled data using clustering. In: Proc. ACM CSS Workshop on Data Mining Applied to Security. (2001)

3. Eskin, E., Arnold, A., Prerau, M., Portnoy, L., Stolfo, S.: A geometric framework for unsupervised anomaly detection: detecting intrusions in unlabeled data. In: Applications of Data Mining in Computer Security. Kluwer (2002)
4. Lazarevic, A., Ertoz, L., Kumar, V., Ozgur, A., Srivastava, J.: A comparative study of anomaly detection schemes in network intrusion detection,. In: Proc. SIAM Conf. Data Mining. (2003)
5. Laskov, P., Schäfer, C., Kotenko, I.: Intrusion detection in unlabeled data with quarter-sphere support vector machines. In: Proc. DIMVA. (2004) 71–82
6. Laskov, P., Schäfer, C., Kotenko, I., Müller, K.R.: Intrusion detection in unlabeled data with quarter-sphere support vector machines (extended version). Praxis der Informationsverarbeitung und Kommunikation **27** (2004) 228–236
7. Ghosh, A.K., Schwartzbard, A., Schatz, M.: Learning program behavior profiles for intrusion detection. In: Proc. of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring, Santa Clara, USA (1999) 51–62 http://www.cigital.com/papers/download/usenix_id99.pdf.
8. Warrender, C., Forrest, S., Perlmutter, B.: Detecting intrusions using system calls: alternative data methods. In: Proc. IEEE Symposium on Security and Privacy. (1999) 133–145
9. Mukkamala, S., Janoski, G., Sung, A.: Intrusion detection using neural networks and support vector machines. In: Proceedings of IEEE Internation Joint Conference on Neural Networks. (2002) 1702–1707
10. Lee, W., Stolfo, S., Mok, K.: A data mining framework for building intrusion detection models. In: Proc. IEEE Symposium on Security and Privacy. (1999) 120–132
11. Stolfo, S.J., Wei, F., Lee, W., Prodromidis, A., Chan, P.K.: KDD Cup - knowledge discovery and data mining competition (1999) http://kdd.ics.uci.edu/databases/kddcup99/kddcup99.html.
12. Lippmann, R., Cunningham, R.K., Fried, D.J., Kendall, K.R., Webster, S.E., Zissman, M.A.: Results of the DARPA 1998 offline intrusion detection evaluation. In: Proc. RAID 1999. (1999) http://www.ll.mit.edu/IST/ideval/pubs/1999/RAID_1999a.pdf.
13. Lippmann, R., Haines, J.W., Fried, D.J., Korba, J., Das, K.: The 1999 DARPA off-line intrusion detection evaluation. Computer Networks **34** (2000) 579–595
14. Lee, W., Stolfo, S.: A framework for constructing features and models for intrusion detection systems. In: ACM Transactions on Information and System Security. Volume 3. (2001) 227–261
15. Riedmiller, M., Braun, H.: RPROP - a fast adaptive learning algorithm. Universität Karlsruhe, Deutschland (1992)
16. Rojas, R.: Neural Networks: A Systematic Approach. Springer-Verlag, Berlin, Deutschland (1996)
17. Friedman, J.: Regularized discriminant analysis. Journal of the American Statistical Association **84** (1989) 165–175
18. Harmeling, S., Dornhege, G., Tax, D., Meinecke, F., Müller, K.R.: From outliers to prototypes: ordering data. Unpublished manuscript (http://ida.first.fhg.de/∼harmeli/ordering.pdf), submitted. (2004)
19. Duda, R., P.E.Hart, D.G.Stork: Pattern classification. second edn. John Wiley & Sons (2001)

2.3 Summary

An experimental framework is presented in which supervised and unsupervised learning methods are evaluated in an network intrusion detection application. Experimental results show that supervised learning significantly outperforms unsupervised learning methods if test data is drawn from a distribution identical to the training data distribution (i.e. known attack detection scenario). In this scenario, non-linear classification methods such as SVM, multi-layer perceptron and decision tree demonstrated best performance with more than 95% true positive at 2% false positive rate. However, under the second scenario, in presence of unknown attacks in test data, performance of supervised learning methods drops significantly rendering SVM with a detection accuracy of to 82% true positive at 10% false positive rate to outperform all other supervised learning methods tested. Although the detection accuracy of unsupervised learning method under-perform supervised methods, the performance of unsupervised learning methods is not impacted by either of the two scenarios. The graph-based γ-outlier detection method which defines the distance of a data point to its k nearest neighbors as anomaly score outperforms all other tested unsupervised methods and compares to the best supervised method with a detection accuracy of 79% at 10% false positives in the unknown attack detection scenario.

CHAPTER 3

Cyber-Critical Infrastructure Protection Using Real-Time Payload-Based Anomaly Detection

3.1 Introduction

While Chapter 2 focuses on the investigation of supervised and unsupervised learning methods to detect network attacks entangled in network protocol headers, this section investigates to what extent unsupervised machine learning allows for the detection of unknown network attacks in network packet payloads particularly in SCADA network traffic.
Critical infrastructures such as electricity generation and transmission, oil & gas production and distribution or water supply heavily rely on SCADA networks for process automation. Given the increasing demand for inter-connectivity and the adoption of standardized network protocols, cyber-critical infrastructures are exposed to a plethora of cyber threats. Known attacks on critical infrastructure such as "Stuxnet" or "BlackEnergy" demonstrated the severe impact advanced cyber-physical attacks can have. These attacks showed the importance of effective network-level protection to detect exploits targeting "Zero-day" vulnerabilities on IT systems to prevent hijacking or denial of service of critical infrastructures. Moreover, patching of known vulnerabilities in industrial control systems is not always feasible due to unknown impact of a patch on the system's integrity.
This contribution proposes a fast, centroid-based anomaly detection method that detects unknown zero-day network attacks on SCADA network protocols based on language models and various distances. Main objective of this contribution is to investigate to what extent the proposed method is capable to not only detect attacks delivered via plain-text protocols such as HTTP but more specifically to evaluate attack detection capabilities in binary application level network protocols typical for SCADA networks such as Remote Procedure Call (RPC), Server Message Block (SMB), or Netbios. To this end, experiments are conducted over network traffic recorded in an industrial automation testbed of a major industrial automation manufacturer. Similarly to experiments in Section 2 attack instances in test data partitions are not contained in training data partitions during cross-validation. Finally, experimental results are compared against the state-of-the-art anomaly detection method *PAYL* [46] which uses Mahalanobis distance over single-byte distributions to perform anomaly detection.

3.2 Publication

Cyber-Critical Infrastructure Protection Using Real-time Payload-based Anomaly Detection *

Patrick Düssel[1], Christian Gehl[1], Pavel Laskov[1,2]
Jens-Uwe Bußer[3], Christof Störmann[3] and Jan Kästner[4]

[1] Fraunhofer Institute FIRST
Intelligent Data Analysis, Berlin 12489, Germany
[2] University of Tübingen
Wilhelm-Schickard-Institute for Computer Science, Tübingen, Germany
[3] Siemens AG
Corporate Technology, Information and Communications, München, Germany
[4] Siemens AG
Industrial Automation Systems, Research & Development, Karlsruhe, Germany

Abstract. With an increasing demand of inter-connectivity and protocol standardization modern cyber-critical infrastructures are exposed to a multitude of serious threats that may give rise to severe damage for life and assets without the implementation of proper safeguards. Thus, we propose a method that is capable to reliably detect unknown, exploit-based attacks on cyber-critical infrastructures carried out over the network. We illustrate the effectiveness of the proposed method by conducting experiments on network traffic that can be found in modern industrial control systems. Moreover, we provide results of a throughput measuring which demonstrate the real-time capabilities of our system.

1 Introduction

Industrial control systems such as supervisory control and data acquisition systems (*SCADA*), distributed control systems (*DCS*), and energy distribution systems (*EDS*) are used to monitor and control industrial automation processes and have been successfully deployed in critical infrastructures including power plants, power and water distribution, and traffic systems. Over the last decade considerable effort has been done to protect computer networks. However, a comparable small amount of research has been dedicated to cyber-security related aspects of critical infrastructure protection mainly because control systems were based on proprietary protocols and separated from public networks.

*This work was supported by the German Bundesministerium für Bildung und Forschung (BMBF) under the project ReMIND (FKZ 01-IS07007A).

Nowadays, with the increasing demand of inter-connectivity and the ongoing convergence towards standardized network protocols communication is realized using well-known transport-layer protocols such as TCP/IP. Consequently, the risk of becoming exposed to novel threats is raised. As availability is most eminent for real-time systems countermeasures to ensure integrity and confidentiality of communication can be barely applied as they usually come along with a reduction of availability. With regard to the strong utilization of computer networks and the increasing transparency of communication software patching becomes not only a very important but also a difficult task. Unfortunately, patching is not always an option since it usually requires a reboot during rare maintainance intervals or, even worse, if it cannot be guaranteed that the patch does not alter the system behavior. As a consequence, critical services in control systems remain vulnerable for a long period of time which cannot always be compensated by existing technical or administrative controls.

Thus, in order to provide adequate protection of process control networks reliable and fast intrusion detection (IDS) is crucial. Intrusion detection methods can be broadly categorized into *signature detection* and *anomaly detection*. While signature detection identifies attacks based on known attack characteristics, anomaly detection tags suspicious events by measuring a deviation from a model of normality. Signature-based intrusion detection systems possess a number of mechanisms for analyzing application-level content, ranging from simple scanning of payload for specific byte patterns, as in Snort [12], to sophisticated protocol analysis coupled with policy scripts, as in Bro [9; 10]. Signature-based IDS typically exhibit a high detection accuracy and is therefore widely deployed as a proper compensating control in enterprise networks. However, the major drawback of signature-based IDS is their reliance on the availability of appropriate exploit signatures. Unfortunately, the rapid development of new exploits and their growing variability make keeping signatures up-to-date a daunting if not impossible task. This motivates investigation of alternative techniques, such as anomaly detection, that are in principle capable to detect unknown attacks. In our contribution we propose a method that is capable to reliably detect unknown, vulnerability-based attacks against industrial control systems that originate from both trusted and untrusted networks. We illustrate the effectiveness of the proposed method by conducting experiments on network traffic that can be typically found (*SCADA*) systems.

The paper is structured as follows. Related work on anomaly detection in SCADA networks is presented in Section 2. In Section 3 we present a topology of a modern SCADA network and briefly describe two attack scenarios that are addressed in our experiments. Details on the architecture of our anomaly detection system can be found in Section 4. Experimental evaluation on real network traffic is carried out in Section 5 which also provides a performance evaluation of our approach. Finally, conclusions are presented in Section 6.

2 Related Work

Anomaly-based IDS have traditionally focused on features derived from network and transport layer protocols. An example of such features can be found in the data mining approach of Lee and Stolfo [7], containing packet, connection and time window features derived from IP and TCP headers. The same work has pioneered the use of "content" features that comprised selected application-level properties such as number shell prompts, number of failed login prompts, etc. deemed to be relevant for detection of specific attacks. Similar features comprising selected keywords from application layer protocols have been used by Mahoney and Chan for anomaly detection [8].

General content-based features using payload distribution of specific byte groups have been first proposed by Kruegel et al. [6] in the context of service-specific anomaly detection using separate normality models for different application layer protocols. Full distributions of byte values have been considered by Wang and Stolfo [14], extended to models of various languages that can be defined over byte sequences, e.g. n-grams [11; 13].

With regard to critical infrastructures previous work on anomaly detection has been mainly focused on physical measurement modeling which differs from our work in that we do not learn models over physical processes and also don't assume prior knowledge on protocols used to transfer measurements.

In the work of Bigham et al. [2] the authors propose to learn a *n-gram model* and an *invariant model* from data that is passed around the system. While the n-gram approach is used for the first four bytes of each data reading to determine a model of sign, decimal point position and most significant digits the latter model is used to determine linear dependencies between different data readings which are expressed as invariants. In a continuative work by Jin et al. [5] which specifically addresses anomaly detection in electricity infrastructures the authors extended the set of invariant models by a *value range model* which marks a data reading to be anomalous if its value exceeds a pre-determined threshold. Furthermore, a *bus-zero-sum* model is deployed which tests current inflow and outflow on a bus for equality. Given these models anomaly scores are finally combined in a probabilistic framework to reason about the likelihood of an anomaly given the set of trained models. Clearly, the features used for anomaly detection strongly depend on a particular domain.

A more network-centric approach is suggested by Antonio et al. [3]. They propose a distributed architecture for high-speed intrusion detection which stipulates the deployment of classification techniques to detect suspicious traffic patterns. It differs from our work in that their method requires label information to detect attacks.

3 SCADA Networks

SCADA networks are widely deployed to monitor and control processes that are essential to a modern society. Typically, a SCADA system consists of a

master terminal unit (MTU), a *human machine interface* (HMI) and one or more *remote terminal units* (RTU) which are connected to field devices such as sensors and actuators. Field device data is periodically transfered to the MTU which continuously reassembles an image of the overall ongoing process. The topology of a SCADA network satisfying state-of-the-art security concepts [1] is shown in Fig.1. The network basically consists of three segments which are segregated by firewalls: an untrusted *enterprise control network* (ECN), a semi-trusted *demilitarized zone* (DMZ) and a *process control system* forming a trusted network. The process control system consists of a *control system network* (CSN) and a *process control network* (PCN). The CSN as the lowest layer network is made up of *automation stations* (AS) which are RTUs capable to exchange field device data via Industrial Ethernet to satisfy real-time requirements. The PCN which is located above the CSN constitutes the most critical part of a SCADA system. It contains a MTU referred to as *operator station* (OS) server and an *engineering station* (ES). While the OS server controls field devices attached to the CSN the engineering station provides an interface for the configuration of systems in both CSN and PCN (e.g. programming of individual AS). Communication between trusted networks is carried out via secure tunnels (e.g. VPN). The DMZ which separates untrusted and trusted networks provides a service interface to untrusted networks.

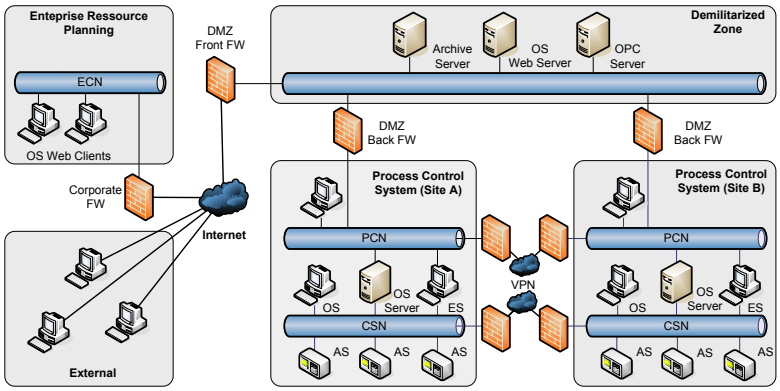


Fig. 1. General topology of a SCADA system

Penetration of a SCADA system requires previous exploitation of existing software vulnerabilities. Generally, there are two kinds of threat scenarios that we address in our contribution:

– ***External Threat***. SCADA systems increasingly provide interfaces to untrusted networks such as corporate networks or the Internet. A threat agent perpetrates a device located in the DMZ (e.g. web server) from an external network either to prevent the system from being accessed by others or to carry out relayed attacks against the process control system.

– ***Internal Threat***. A threat agent with direct access to the process control system (CSN and PCN) located at one of the facility sites mounts exploit-based attacks against critical services in the network. This threat becomes particularly serious in the presence of unmanned stations and wireless communication.

4 Methodology

The key benefit of payload-based anomaly detection lies in its ability to cope with unknown attacks. The following four stages outline the essential building blocks of our approach and will be explained in detail for the rest of this section.

1. *Network Sensor*. Inbound transport layer packets are captured from the network by *Bro*[5] which provides fast and robust TCP re-assembly. TCP Payload is extracted and forwarded to the feature extraction stage.
2. *Feature Extraction*. Byte sequences are mapped into a feature space which is defined by the set of sequential features extracted from incoming sequences. The utilization of efficient data structures allows to operate in high-dimensional feature spaces. Details on the feature extraction process can be found in Section 4.2.
3. *Similarity Computation*. A proper definition of similarity between byte sequences is crucial for payload-based anomaly detection. The similarity between byte sequences is determined by computing the pairwise distance between their respective vectorial representation. The similarity computation is explained in Section 4.3.
4. *Anomaly Detection*. The anomaly detector initially learns a global model of "normality" which is represented by the center of mass of training data points. At detection time arriving byte sequences are compared to the previously learned model and based on a distance an anomaly score is calculated. The anomaly detection process is described in Section 4.4.

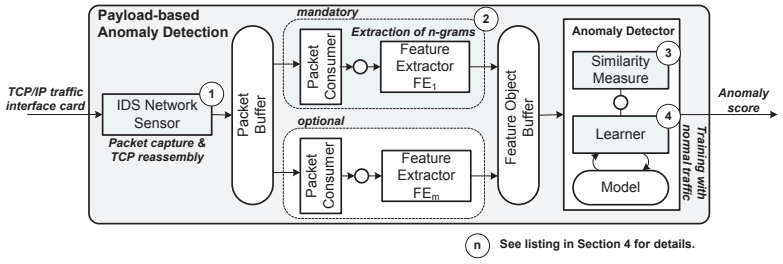


Fig. 2. Payload-based anomaly detection system

[5]See http://www.bro-ids.org/ for details.

4.1 Network Sensor

Bro as the basis of our prototypical implementation is a Unix-based Network Intrusion Detection System that passively monitors network traffic. To match our requirements as a TCP/IP network sensor we exclude parsing of network traffic and signature matching. The Berkeley Packet Filters (BPF), also known as tcpdump expressions, provided by *Bro* can be used in a highly flexible and adapted mode to process the desired packets for detection. Moreover, *Bro* takes care of fragmented packets and TCP re-assembly which are important factors for robust processing of TCP network traffic. Byte sequences are extracted from the payload of incoming packets and forwarded to the feature extraction stage.

4.2 Feature Extraction

Anomaly detection usually requires data to be in a vectorial representation. Therefore, the feature extraction process embeds a byte sequence s into a feature space $\mathcal{F}$ in which similarity between sequences can be computed. By moving a sliding window of a particular length n over a sequence s a set of unique, sequential features – so called n-grams – is extracted. The resulting feature space is defined over the set $I \subseteq \Sigma^n$ of possible n-grams u induced by an alphabet Σ:

$$\phi(s) \mapsto (\phi_u(s))_{u \in I} \in \mathcal{F}, \;\; u \in \Sigma^n \tag{1}$$

Once a sequence is embedded into $\mathcal{F}$ various feature maps ϕ can be applied. Commonly used feature maps for *contiguous strings* are explained below:

- *Count Embedding.* The value of the coordinate $\phi_u^{cnt}(s)$ reflects the count of a string u contained in s
- *Frequency Embedding.* The value of the coordinate $\phi_u^{freq}(s)$ reflects the term frequency of a string u contained in s. Essentially, this mapping corresponds to a *count embedding* normalized by the maximum number of n-grams contained in s.
- *Binary Embedding.* The value of the coordinate $\phi_u^{bin}(s)$ reflects the presence of a string u in s.

4.3 Similarity Measure

The utilization of a geometric representation of a byte sequence through ϕ allows to deploy classical, vector-based similarity measures such as distance functions. A list of relevant function is provided in Table 1.

4.4 Anomaly Detection

Anomalies can be considered as deviations from a previously learned model of normality which is represented as the center of mass of a set of data points. To this end, the center of mass $\mathbf{c}$ can be defined by:

$$\mathbf{c} = \frac{1}{\ell} \sum_{i=1}^{\ell} \phi(x_i), \; \phi(x_i) \in \; \mathbb{R}^n \tag{2}$$

Name	Similarity	Name	Similarity
Manhattan	$\sum_{i=1}^{n} \lvert x_i - y_i \rvert$	Canberra	$\sum_{i=1}^{n} \frac{\lvert x_i - y_i \rvert}{x_i + y_i}$
Euclidean	$\sum_{i=1}^{n} (x_i - y_i)^2$	Chi-Squared	$\sum_{i=1}^{n} \frac{(x_i - y_i)^2}{x_i + y_i}$

Table 1. Similarity measures

where $\phi(x_i)$ refers to a training point explicitly embedded in a n-dimensional geometric space as described in Section 4.2. In order to down-weight sparse representations $\mathbf{c}$ is normalized as follows:

$$\hat{\mathbf{c}} = \frac{\mathbf{c} \cdot \mathbf{w}}{\|\mathbf{w}\|_1} \tag{3}$$

where $\mathbf{w}$ denotes the standard deviation of individual dimensions in $\mathcal{F}$ observed over ℓ training points. Finally, an anomaly score S_x for an unknown data point x is computed by calculating the distance to the center of mass of training data:

$$S_x = \sum_{j=1}^{n} d(\hat{\mathbf{c}}_j, \phi_j(x)) \tag{4}$$

A similar anomaly detection method is referred to as *Payl* [14] which relies on the computation of a *simplified Mahalanobis distance*. The anomaly score S_x is defined as the variance-scaled distance from an unknown data point x to the center of mass $\mathbf{c}$:

$$S_x = \sum_{j=1}^{n} \frac{d(\mathbf{c_j}, \phi_j(x))}{\sigma_j} \tag{5}$$

5 Experiments

With regard to the attack scenarios outlined in Section 3 we evaluate our method on two data sets containing traffic monitored at our institute as well as in a SCADA testbed. For each of the two data sets we recorded a collection of service-specific attacks including buffer overflows which can be used for *privilege escalation* as well as web application attacks. Exploits were taken from the Metasploit framework [6] and from common security forums such as *securityfocus.com* and *remote-exploit.org*.

The first data set (***Web07***) contains inbound HTTP traffic captured in a DMZ over a period of five days and consists of approx. 1,000,000 TCP packets. The corresponding attack set comprises 42 attack instances exploiting 11 different vulnerabilities in HTTP services together with a *Nessus* vulnerability scan. The list of HTTP-related exploits can be found in Table 2.

[6] http://www.metasploit.com

	Id	CVE	Description	Type	Id	CVE	Description	Type
HTTP	1	2002-0071	IIS (ISAPI/HTR Script)	Buf	7	2006-5478	Novell eDirectory	Buf
	2	2001-0500[1]	IIS (ISAPI/Indexing)	Buf	8	2003-1192	IA WebMail	Buf
	3	2001-0241	IIS (ISAPI/Printing)	Buf	9	2000-0884	IIS Dir. Traversal	Web
	4	2004-1134	IIS (ISAPI/W3Who)	Buf	10	2006-2644	AwStats (Logging)	Web
	5	2003-0109	IIS (NTDLL/WebDAV)	Buf	11	2005-2847	Barracuda (Spam)	Web
	6	2002-2270	Alt-N WebAdmin	Buf	12	-	Nessus scan	Web
RPC	1	2003-0352[2]	RPC (DCOM)	Buf	6	2006-4696	SMB Mailslot	Buf
	2	2003-0533	LSASS	Buf	7	2006-3441	RPC (DNS)	Buf
	3	2004-1080	RPC (WINS)	Buf	8	2008-4250[3]	SMB (SRVSVC)	Buf
	4	2005-0059	RPC (MSMQ)	Buf	9	-	RPC scan	-
	5	2006-3439	SMB (SRVSVC)	Buf				

Table 2. Attack sets ([1] *Code Red*, [2] *W32.Blaster*, [3] *Conficker*)

The data sets ***Aut09a*** (57100 TCP packets) and ***Aut09b*** (765,103 TCP packets) were captured in a process control network and contain payload of binary application-layer protocols such as SMB and RPC and Netbios.
The corresponding attack set contains 19 attack instances exploiting eight different vulnerabilities. Attacks were carried out using various attack payloads (i.e. account creation, (reverse) shell binding and VNC server injection. RPC-related exploit details are provided in Table 2.

5.1 Experimental Setup

In order to find a model that maximizes the detection performance a validation phase precedes the actual evaluation of our method. It is important to mention that *data used during validation is not employed during evaluation*. To this end, data is split into three distinct partitions for training, validation and testing from which samples are randomly drawn. Each sample consists of 1000 TCP payloads. Model selection is implemented as a 10-fold cross validation. Both, validation and test samples are equally mixed with **distinct** subsets of available attack classes. Each model is trained on a training sample and subsequently validated on ten distinct validation samples. Finally, the model that maximizes the average detection accuracy during validation is applied on a test sample. The detection accuracy is measured in terms of area under *receiver operating characteristic* curve ($AUC_{0.01}$) which integrates true positive values over the false positive interval [0,0.01]. For statistical reasons experiments on both data sets are repeated over 20 repetitions.

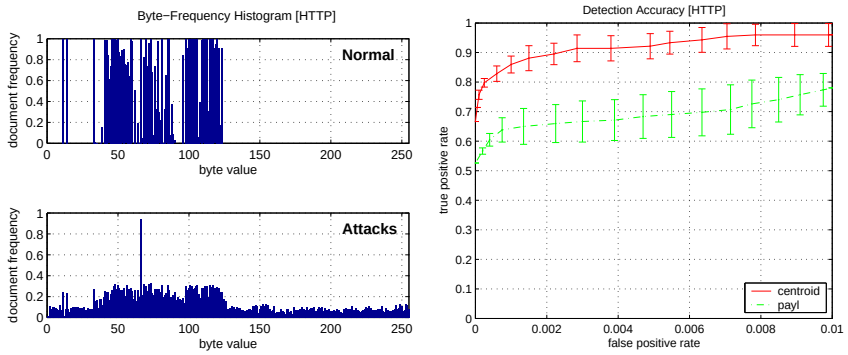
5.2 Experiments on HTTP Traffic

In this experiment we investigate the detection of overflow-based attacks and web application attacks carried out over the well-known HTTP protocol. As shown in Table 3 cross validation reveals that variance-weighted Manhattan distance over 3-grams is chosen to be the best model over all repetitions. Considering the average AUC values for the best three models the choice of the feature embedding has only a minor impact on the detection accuracy. As shown in Fig. 3(b) with a

Measure	Weighting	n-gram	Embedding	AUC_{avg}	AUC_{std}	$\|Model_{best}\|$
Manhattan	variance	3	freq	0.9231	±0.0370	10
Manhattan	variance	4	freq	0.9169	±0.0406	2
Canberra	none	4	bin	0.9091	±0.0395	5

Table 3. $AUC_{0.01}$ values of selected models in validation phase

detection rate of 88% at a false alarm rate of 0.2% our method outperforms *Payl*. A detailed analysis of the cost associated with the detection of individual attack



(a) Byte histogram (b) Detection Accuracy

Fig. 3. Detection of unknown attacks in HTTP traffic

classes is given in Table 4. It shows that overflow-based attacks are perfectly detected while some web application attacks suffer from a small number of false positives.

Class	Instances	Centroid		Payl	
		FP_{avg}	FP_{std}	FP_{avg}	FP_{std}
1 - 8	25	0.0000	±0.0000	0.0000	±0.0000
9	3	0.0008	±0.0013	0.0060	±0.0014
10	5	0.0072	±0.0079	0.0360	±0.0128
11	6	0.0045	±0.0053	0.0260	±0.0081
12	3	0.0056	±0.0037	0.0264	±0.0075

Table 4. False positive rate per HTTP attack class on test data

This is not a surprise considering the significant differences in the byte histograms of normal data and overflow-based attacks which are compared in Fig.3(a). Unlike overflow-based attacks which heavily rely on the utilization of bytes in the range between **0x7f** and **0xff** to carry malicious opcode web application attacks usually exploit vulnerabilities of scripts using bytes that are commonly found in normal HTTP traffic. This explains the comparably high cost associated with the detection of this type of attacks. However, the detection

accuracy for web application attacks can be further enhanced by incorporating syntactic information into sequential feature representations [4].

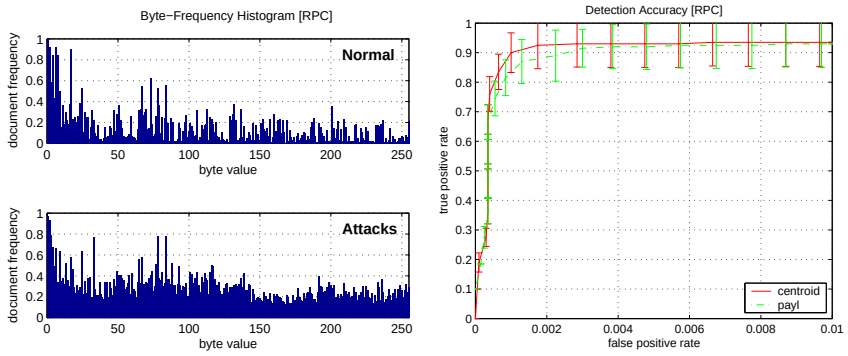
5.3 Experiments on RPC Traffic

While in the previous section we address the problem of detecting attacks carried out over text-based application protocols such as HTTP in this section we investigate the detection of overflow-based attacks carried out over binary protocols such as Netbios, SMB and RPC. Experiments were applied to the ***Aut09a*** data set. As shown in Table 5 cross validation chooses a variance-weighted Chi-squared distance using a frequency embedding as the best model.

Measure	Weighting	n-gram	Embedding	AUC_{avg}	AUC_{std}	$\lvert Model_{best} \rvert$
Chi-Squared	variance	4	freq	0.8655	$\pm$0.0721	13
Canberra	none	4	bin	0.8548	$\pm$0.0892	3
Canberra	none	2	bin	0.8459	$\pm$0.0911	3

Table 5. $AUC_{0.01}$ values of selected models during validation

Similarly to the results of experiments on HTTP in this experiment our method attains a detection rate of 92% at a false positive rate of 0.2%. Interestingly, although a rather complex model (i.e. Chi-squared distance over 4-grams) is chosen our method demonstrates a marginal improvement in detection accuracy only compared to the results obtained using *Payl*. However, while most of the attacks



(a) Byte histogram (b) Detection Accuracy

Fig. 4. Detection of unknown attacks in RPC/SMB traffic

are perfectly separable from the normal data (cf. Table 6) "SMB-Mailslot" is the only attack that suffers a comparably high false positive rate of approx. 5%. This particular denial of service attack exploits a vulnerability in the Microsoft server service (SVR.SYS) and is triggered by a specially crafted but fairly short malformed SMB packet.

Class	Instances	Centroid		Payl	
		FP_{avg}	FP_{std}	FP_{avg}	FP_{std}
1	3	0.0004	±0.0005	0.0004	±0.0005
2	1	0.0003	±0.0006	0.0003	±0.0006
3	1	0.0001	±0.0003	0.0003	±0.0005
4	1	0.0000	±0.0000	0.0000	±0.0000
5	6	0.0003	±0.0005	0.0003	±0.0005
6	1	0.0705	±0.0684	0.2783	±0.0464
7	1	0.0005	±0.0008	0.0000	±0.0000
8	4	0.0005	±0.0008	0.0018	±0.0014
9	1	0.0950	±0.1118	0.0231	±0.0097

Table 6. False positive rate per RPC/SMB attack class on test data

5.4 Performance Evaluation

The performance of our prototypical implementation is tested inside of a virtual network which consists of two client machines[7] which operates as sender and receiver of network traffic. Each client is installed on a separate host[8]. To simulate a proper online scenario we use the tcpreplay tool suite[9] to replay the captured tcpdump files between both clients. In order to maintain the original transport-layer characteristics (i.e. to preserve the TCP connection on the receiver site) the Media Access Control address of all packets are rewritten to the receivers's interface card. Thus, the prototype can process full TCP connections, as it is required in a production environment. Table 7 shows the throughput in MBits per second obtained from running our prototype in single CPU mode. During measurement anomaly detection is applied to incoming traffic. The performance evaluation reveals that the throughput of our prototype depends on the ratio of inbound packets as well as the n-gram size at hand. The drop of performance linked with the increasing n-gram size is due to the fact that the dimensionality of the underlying feature space increases exponentially which also affects the feature extraction process.

Data set	Replayed packets	Analyzed packets	1-gram	2-gram	3-gram	4-gram
Web07	1,000,000	6.0%	429.1Mbps	348.6Mbps	309.9Mbps	253.1Mbps
Aut09b	765,103	52.3%	197.5Mbps	113.1Mbps	31.4Mbps	18.3Mbps

Table 7. Throughput of the prototype using a single CPU with the anomaly detector analyzing incoming traffic.

6 Conclusion

In this contribution we propose a fast and effective payload-based anomaly detection system which can be deployed in cyber-critical infrastructures to reliably

[7] FreeBSD images using 2 CPUs with 4 GByte RAM memory. The BPF buffer size is set to 10MByte.

[8] The server hardware are two Sun Fire X4100 M2 with 4 CPU x 2,8Ghz.

[9] See http://tcpreplay.synfin.net/trac/ for details.

detect zero-day attacks. Our method relies on the computation of similarity between transport-layer packet payloads embedded in a geometric space. We carry out experiments on traffic containing various application-layer protocols (text-based and binary) that were captured in different network segments of a SCADA testbed. With a detection rate of **88%-92%** at a false positive level of **0.2**% the method has been proved to be useful for the detection of unknown attacks in network traffic. Considering the nature of critical infrastructures future work should address intelligent incident handling which includes the incorporation of process model features into anomaly detection.

Bibliography

[1] Security concept pcs7 and wincc-basic document. White paper, Siemens AG, April 2008. A5E02128732-01.

[2] J. Bigham, D. Gamez, and N. Lu. Safeguarding scada systems with anomaly detection. In *MMM-ACNS*, pages 171–182, 2003.

[3] S. D'Antonio, F. Oliviero, and R. Setola. High-speed intrusion detection in support of critical infrastructure protection. In *Proc. 1st International Workshop on Critical Information Infrastructures Security*, 2006.

[4] P. Düssel, C. Gehl, P. Laskov, and K. Rieck. Incorporation of application layer protocol syntax into anomaly detection. In *Proc. of International Conference on Information Systems Security (ICISS)*, pages 188–202, 2008.

[5] X. Jin, J. Bigham, J. Rodaway, D. Gamez, and C. Phillips. Anomaly detection in electricity cyber infrastructures. In *Proceedings of the International Workshop on Complex Networks and Infrastructure Protection (CNIP-06)*, 2006.

[6] C. Kruegel, T. Toth, and E. Kirda. Service specific anomaly detection for network intrusion detection. In *Proc. of ACM Symposium on Applied Computing*, pages 201–208, 2002.

[7] W. Lee and S. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information Systems Security*, 3:227–261, 2000.

[8] M. Mahoney and P. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 376–385, 2002.

[9] V. Paxson. Bro: a system for detecting network intruders in real-time. In *Proc. of USENIX Security Symposium*, pages 31–51, 1998.

[10] V. Paxson. The bro 0.8 user manual. Lawrence Berkeley National Laboratroy and ICSI Center for Internet Research, 2004.

[11] K. Rieck and P. Laskov. Language models for detection of unknown attacks in network traffic. *Journal in Computer Virology*, 2(4):243–256, 2007.

[12] M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proc. of USENIX Large Installation System Administration Conference LISA*, pages 229–238, 1999.

[13] K. Wang, J. Parekh, and S. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Recent Adances in Intrusion Detection (RAID)*, pages 226–248, 2006.

[14] K. Wang and S. Stolfo. Anomalous payload-based network intrusion detection. In *Recent Adances in Intrusion Detection (RAID)*, pages 203–222, 2004.

3.3 Summary

This contribution proposes a fast and effective centroid-based anomaly detection method which utilizes higher order language models in combination with distance measures to detect payload-based zero-day attacks in SCADA network traffic. The method relies on the computation of similarity between transport-layer network packet payloads embedded in a geometric space. With a detection rate of 88-92% true positives at 0.2% false positives the method has proven to be useful for the detection of unknown attacks in SCADA network traffic. Moreover, experimental results suggest that the proposed method significantly outperforms (88% true positives at 0.2% false positives) the state-of-the-art anomaly detection system *PAYL* (65% true positives at 0.2% false positives) over plain-text protocols. However, the accuracy improvement of the proposed method over binary network protocols is only marginal compared to *PAYL*.

CHAPTER 4

Automatic feature selection for anomaly detection

4.1 Introduction

Learning models in presence of a plethora of available features is a commonly known challenge in machine learning. For example, while Chapter 2 focuses on network packet header features, experiments in Chapter 3 are based on content byte stream features and Chapter 5 discusses the usefulness of structural features to detect network attacks. The choice of the right set of features is crucial not only for learning the right model but also to allow for efficient learning and decision making. In order to investigate which features are most effective for anomaly detection in network security, this section introduces a method for automatic feature selection and provides an experimental evaluation to identify features to maximize accuracy of anomaly-based attack detection in network traffic.
The vast majority of feature selection methods requires label information in order to find discriminative features [e.g. 90]. In presence of feature blending (i.e. learning predictors over a combination of features of different sets) and in absence of label information – typical for anomaly detection problems – a different approach is required. To this end, this contribution proposes a novel method for automated feature selection in anomaly detection which allows to identify a set of discriminative features from different feature sets alongside optimal feature set mixture coefficients which realize a minimal-volume description of data. Experiments are conducted on unsanitized HTTP data for network intrusion detection to investigate the impact of weighting features from different feature sets on the detection accuracy using Support Vector Data Description (SVDD) - a particular implementation of the One-class Support Vector Machine. To this end, two different feature sets are generated (syntactic features and sequential features). Syntactic feature sets are obtained from parsing and tokenizing HTTP requests using the *Binpac* protocol analyzer. For each of the two feature sets, three different types of features are generated: 3-grams/ frequency embedding, 3-grams/ count embedding and expert features which consists of a set of 16 different features describing length, entropy as well as special character presence in an HTTP request or HTTP request token.

4.2 Publication

Automatic Feature Selection for Anomaly Detection

Marius Kloft
Technical University of Berlin
Dept. of Computer Science
Berlin, Germany
kloft@cs.tu-berlin.de

Ulf Brefeld
Technical University of Berlin
Dept. of Computer Science
Berlin, Germany
brefeld@cs.tu-berlin.de

Patrick Düssel
Fraunhofer Institute FIRST
Intelligent Data Analysis
Berlin, Germany
patrick.duessel
@first.fraunhofer.de

Christian Gehl
Fraunhofer Institute FIRST
Intelligent Data Analysis
Berlin, Germany
christian.gehl
@first.fraunhofer.de

Pavel Laskov*
Fraunhofer Institute FIRST
Intelligent Data Analysis
Berlin, Germany
pavel.laskov
@first.fraunhofer.de

ABSTRACT

A frequent problem in anomaly detection is to decide among different feature sets to be used. For example, various features are known in network intrusion detection based on packet headers, content byte streams or application level protocol parsing. A method for automatic feature selection in anomaly detection is proposed which determines optimal mixture coefficients for various sets of features. The method generalizes the support vector data description (SVDD) and can be expressed as a semi-infinite linear program that can be solved with standard techniques. The case of a single feature set can be handled as a particular case of the proposed method. The experimental evaluation of the new method on unsanitized HTTP data demonstrates that detectors using automatically selected features attain competitive performance, while sparing practitioners from a priori decisions on feature sets to be used.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*; I.2.6 [**Artificial Intelligence**]: Learning—*Parameter learning*; I.5.2 [**Pattern Recognition**]: Design Methodology—*Classifier design and evaluation, Feature evaluation and selection*

General Terms

Algorithms, Experimentation, Security

*Pavel Laskov is also affiliated with University of Tübingen, Wilhelm-Schickard-Institute, Germany.

AISec'08, October 27,2008, Alexandria, Virginia, USA.

Keywords

Machine learning, anomaly detection, support vector data description, feature selection, multiple kernel learning, intrusion detection, network security

1. INTRODUCTION

The main merit of anomaly detection techniques is their ability to detect previously unknown attacks. One might think that the collective expertise amassed in the computer security community rules out major outbreaks of "genuinely novel" exploits. Unfortunately, a wide-scale deployment of efficient tools for obfuscation, polymorphic mutation and encryption results in an exploding variability of attacks. Although being only "marginally novel", such attacks quite successfully defeat signature-based detection tools. This reality brings anomaly detection back into the research focus of the security community.

The majority of anomaly detection methods use some form of machine learning techniques to devise a model of normality from observed normal traffic. They may vary in features being used but share the general idea of measuring anomality of new objects by their distance (in some metric space) from the learned model of normality, historically also known as "the sense of self" [2]. Apart from this theoretical observation, in practice the effectiveness of anomaly detection crucially depends on the choice of features. Various features have been deployed for network intrusion detection, such as raw values of IP and TCP protocol headers [6, 7], time and connection windows [5], byte histograms and n-grams [15, 14], and "bag-of-tokens" language models [10, 11]. While packet header based features have been shown to be effective against probes and scans (which many practitioners consider uninteresting anyway), other kinds of attacks, e.g. remote buffer overflows, require more advanced payload processing techniques. The right kind of features for a particular application has always been considered as the matter of a judicious choice (or trial and error).

But what if this decision is really difficult to make? Given the choice of several kinds of features, a poor a priori decision would lead to an inappropriate model of normality being learned. A better strategy is to have a *learning algorithm itself* decide which set of features is the best. The reason

for that is that learning algorithms find models with optimal generalization properties, i.e. the ones that are valid not only for observed data but also for the data to be dealt with in the future. The a priori choice of features may bias the learning process and lead to worse detection performance. By leaving this choice to the learning algorithm, the possibility of such bias is eliminated.

The problem of automatic feature selection has been well studied in the machine learning community in the context of classification, i.e. choosing among two or more labels to be assigned to events [4; 8; 16; 3, e.g.]. The classification setup, however, is hardly appropriate for anomaly detection since the training data contains examples of only one class, the normal traffic. To enable automatic feature selection for anomaly detection, we derive an appropriate formulation for one-class-classification, a particular kind of anomaly detection using support vector data description (SVDD) [13]. Our approach generalizes the vanilla SVDD that is contained as a special case when only a single feature vector is used. The solution to our feature selection problem is a sparse linear combination of features that realizes a minimal-volume description of the data. The underlying optimization can be phrased as a semi-infinite linear program and solved by standard techniques. A further advantage of the proposed method is that it allows training on contaminated data by limiting the impact of single events on the learned model. To emphasize this feature, we have carried out experiments on *unsanitized* training data obtained "from the wire".

Our paper is structured as follows. Section 2 reviews the problem setting of classical one-class anomaly detection with only a single feature mapping. We derive our feature selection SVDD in Section 3 where we also state the final optimization problem. Section 4 reports on empirical results and Section 5 concludes.

2. ONE-CLASS ANOMALY DETECTION

In this section, we briefly review the classical support vector data description (SVDD) [13]. We are given a set of n *normal* inputs $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$ and a function $\phi : \mathcal{X} \to \mathcal{F}$ extracting features out of the inputs. For instance, $\mathbf{x}_i$ may refer to the i-th recorded request and $\phi(\mathbf{x}_i)$ may encode the vector of bigrams occurring in $\mathbf{x}_i$.

The goal in anomaly detection is to find a description of the normal data such that anomalous data can be easily identified as outliers. In our one-class scenario, this translates to finding a minimal enclosing hypersphere (i.e., center $\vec{w}$ and radius R) that contains the normal input data [13]. Given the function

$$f(\mathbf{x}) = \|\phi(\mathbf{x}) - \vec{w}\|^2 - R^2,$$

the boundary of the ball is described by the set $\{\mathbf{x} : f(\mathbf{x}) = 0 \wedge \mathbf{x} \in \mathcal{X}\}$. That is, the parameters of f are to be chosen such that $f(\mathbf{x}) < 0$ for normal data and $f(\mathbf{x}) > 0$ for anomalous points. The center $\vec{w}$ and the radius R can be computed accordingly by solving the following optimization problem [13]

$$\begin{aligned} \min_{\vec{w},R,\vec{\xi}} \quad & R^2 + \eta \sum_i \xi_i \\ \text{s.t.} \quad & \forall_{i=1}^n : \|\phi(\mathbf{x}_i) - \vec{w}\|^2 \leq R^2 + \xi_i \\ & \forall_{i=1}^n : \xi_i \geq 0. \end{aligned}$$

The trade-off parameter $\eta > 0$ adjusts point-wise violations of the hypersphere. That is, a concise description of the data might benefit from omitting some data points in the computation of the solution. Discarded data points induce slack that is absorbed by variables ξ_i. Thus, in the limit $\eta \to \infty$, the hypersphere will contain all input examples irrespectively of their utility for the model and $\eta \to 0$ implies $R \to 0$ and the center $\vec{w}$ reduces to the centroid of the data. In general, model selection strategies such as cross-validation are necessary not only to find optimal user-defined parameters such as the trade-off η, but also to choose an appropriate feature representation ϕ. In the next section, we detail an approach to automatically select the optimal linear combination of several feature mappings.

3. AUTOMATIC FEATURE SELECTION FOR ANOMALY DETECTION

In this section, we present our approach to automatic feature selection for anomaly detection. Our approach generalizes the support vector data description (SVDD) [13] that is obtained as a special case when only a single feature mapping is given. In contrast to the previous section we are now given k feature mappings $\phi_1, \ldots, \phi_k$ with $\phi_j : \mathcal{X} \to \mathcal{F}_j$, $1 \leq j \leq k$, in addition to the n input instances $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$. For instance, $\mathbf{x}_i$ may refer to the i-th recorded request and $\phi_j(\mathbf{x}_i)$ may encode the j-gram feature vector of $\mathbf{x}_i$.

Besides finding a center and radius, the operational goal is now to learn a linear combination of the given feature mappings to realize the minimal model. This can be expressed equivalently as an embedding of $\phi_1, \ldots, \phi_k$ with mixture coefficients $\beta_1, \ldots, \beta_k$. That is, the model f is now given by

$$f(\mathbf{x}) = \left\| \begin{pmatrix} \sqrt{\beta_1}\phi_1(\mathbf{x}) \\ \vdots \\ \sqrt{\beta_k}\phi_k(\mathbf{x}) \end{pmatrix} - \begin{pmatrix} \vec{w}_1 \\ \vdots \\ \vec{w}_k \end{pmatrix} \right\|^2 - R^2$$

and the above SVDD optimization problem can be generalized accordingly to multiple feature mappings. In the following we write $\vec{w} = (\vec{w}_1, \ldots, \vec{w}_k)^\mathsf{T}$ to avoid cluttering the notation unnecessarily. We are now ready to state the *primal* optimization problem for one-class anomaly detection with multiple feature mappings.

Optimization Problem 1 (primal). *Given n instances $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$, k feature mappings $\phi_1, \ldots, \phi_k$ with $\phi_j : \mathcal{X} \to \mathcal{F}_j$, and $\eta > 0$. The primal feature selection SVDD optimization problem is given by*

$$\begin{aligned} \min_{\vec{w},R,\vec{\xi},\vec{\beta}} \quad & R^2 + \eta \sum_{i=1}^n \xi_i \\ \text{s.t.} \quad & \forall_{i=1}^n : \quad \|\psi_\beta(\mathbf{x}_i) - \vec{w}\|^2 \leq R^2 + \xi_i \\ & \forall_{i=1}^n : \xi_i \geq 0 \\ & \forall_{j=1}^k : \beta_j \geq 0 \\ & \sum_{i=1}^k \beta_j = 1, \end{aligned} \tag{1}$$

where $\psi_\beta(\mathbf{x}_i) = (\sqrt{\beta_1}\phi_1(\mathbf{x}_i), \ldots, \sqrt{\beta_k}\phi_k(\mathbf{x}_i))^\mathsf{T}$.

The last constraint in Optimization Problem 1 requires the mixing coefficients to sum to one which corresponds to an L_1 regularization. We thus promote sparsity and aim at selecting subsets of the k feature mappings. From a geometrical

point-of-view, Optimization Problem 1 can be understood as follows: Redundant and deceptive feature mappings lead to arbitrary and widespread data representations and thus render concise spherical descriptions impossible. Such inappropriate embeddings will be penalized by vanishing mixing coefficients β_j. On the contrary, useful feature mappings are promoted during the optimization, hence enforcing concise data descriptions.

Unfortunately, we cannot solve Optimization Problem 1 directly since it is not convex due to non-linear dependencies between $\vec{\beta}$ and $\vec{w}$, which we see by expanding the term

$$\|\psi_\beta(\mathbf{x}_i) - \vec{w}\|^2 = \sum_{j=1}^{k} \beta_j \langle \phi_j(\mathbf{x}_i), \phi_j(\mathbf{x}_i) \rangle \tag{2a}$$

$$- 2 \sum_{j=1}^{k} \langle \sqrt{\beta_j} \phi_j(\mathbf{x}_i), \vec{w}_j \rangle + \langle \vec{w}, \vec{w} \rangle. \tag{2b}$$

Moreover, setting $\phi_j(\mathbf{x}) = \vec{0}$ for $1 \leq j \leq k$, Equation (1) can be solved for the radius R which can be expressed in terms of the center $\vec{w}$ and a non-negative offset ϵ^2,

$$\|\vec{0} - \vec{w}\|^2 \leq R^2 + \xi_i \quad \Rightarrow \quad R^2 = \langle \vec{w}, \vec{w} \rangle + \epsilon^2. \tag{3}$$

A remedy to the nonlinearity in $\vec{w}$ and $\vec{\beta}$ is a variable substitution by $\vec{v}_j = \sqrt{\beta_j} \vec{w}_j$. Together with Equations (2) and (3) we obtain a convex analogue of the optimization problem (1) given by

$$\begin{aligned}
\min_{\vec{v}, \epsilon, \vec{\xi}, \vec{\beta}} \quad & \epsilon^2 + \sum_{j=1}^{k} \frac{1}{\beta_j} \langle \vec{v}_j, \vec{v}_j \rangle + \eta \sum_{i=1}^{n} \xi_i \\
\text{s.t.} \quad & \forall_{i=1}^{n} : \ \epsilon^2 + \xi_i \geq \sum_{j=1}^{k} \beta_j \langle \phi_j(\mathbf{x}_i), \phi_j(\mathbf{x}_i) \rangle \\
& \qquad - 2 \sum_{j=1}^{k} \langle \phi_j(\mathbf{x}_i), \vec{v}_j \rangle \\
& \forall_{i=1}^{n} : \ \xi_i \geq 0 \,; \quad \forall_{j=1}^{k} : \ \beta_j \geq 0 \,; \quad \sum_{j=1}^{k} \beta_j = 1.
\end{aligned}$$

The above optimization problem is convex and has only linear constraints that can now be integrated into the objective by the Lagrange Theorem. For any valid $\vec{\beta} \in \{\vec{\beta}' : \sum_j \beta'_j = 1 \wedge \beta'_j \geq 0\}$ we obtain a partial Lagrangian by introducing nonnegative Lagrange multipliers $\vec{\alpha}, \vec{\mu} \geq 0$, leading to the Lagrangian L that needs to be minimized.

$$\begin{aligned}
L(\vec{v}, \epsilon, \vec{\xi}, \vec{\beta}, \vec{\alpha}, \vec{\mu}) = \epsilon^2 & + \sum_{j=1}^{k} \frac{1}{\beta_j} \langle \vec{v}_j, \vec{v}_j \rangle + \eta \sum_{i=1}^{n} \xi_i - \sum_{i=1}^{n} \mu_i \xi_i \\
& - \sum_{i=1}^{n} \alpha_i \left(- \sum_{j=1}^{k} \beta_j \langle \phi_j(\mathbf{x}_i), \phi_j(\mathbf{x}_i) \rangle \right. \\
& \qquad \left. + 2 \sum_{j=1}^{k} \langle \phi_j(\mathbf{x}_i), \vec{v}_j \rangle + \epsilon^2 + \xi_i \right).
\end{aligned}$$

The Lagrangian reaches its minimal value when it is minimized with respect to the primal variables $\vec{v}, \epsilon, \vec{\beta}, \vec{\xi}$ and maximized with respect to the Lagrange multipliers; hence, the optimum is found at a saddle-point. Setting the partial derivatives with respect to the primal variables $\epsilon, \vec{v}$, and $\vec{\xi}$ to zero yields

$$\frac{\delta L}{\delta \epsilon} \stackrel{!}{=} 0 \quad \Rightarrow \quad \sum_{i=1}^{n} \alpha_i = 1 \tag{4a}$$

$$\frac{\delta L}{\delta \vec{v}} \stackrel{!}{=} 0 \quad \Rightarrow \quad \vec{v}_j = \beta_j \sum_{i=1}^{n} \alpha_i \phi_j(\mathbf{x}_i), \quad 1 \leq j \leq k \tag{4b}$$

$$\frac{\delta L}{\delta \vec{\xi}} \stackrel{!}{=} 0 \quad \Rightarrow \quad \eta - \mu_i - \alpha_i = 0, \qquad 1 \leq i \leq n. \tag{4c}$$

Equation (4c) together with the nonnegativity constraints on α_i and μ_i leads to the so-called box-constraints $0 \leq \alpha_i \leq \eta$. Resubstitution of Equations (4) into the primal Lagrangian removes its dependence on the primal variables:

$$L(\alpha) = \sum_{i=1}^{n} \alpha_i \sum_{j=1}^{k} \beta_j K_j(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i,\ell=1}^{n} \alpha_i \alpha_\ell \sum_{j=1}^{k} \beta_j K_j(\mathbf{x}_i, \mathbf{x}_\ell).$$

Together with the minimization over $\vec{\beta}$ we resolve the following min-max problem

$$\begin{aligned}
\min_{\vec{\beta}} \ \max_{\vec{\alpha}} \quad & L(\alpha) \\
\text{s.t.} \quad & \forall_{i=1}^{n} : \ 0 \leq \alpha_i \leq \eta \,; \quad \sum_{i=1}^{n} \alpha_i = 1 \\
& \forall_{j=1}^{k} : \ \beta_j \geq 0 \,; \quad \sum_{j=1}^{k} \beta_j = 1,
\end{aligned} \tag{5}$$

where we introduce kernel $K_j(\mathbf{x}, \mathbf{x}') = \langle \phi_j(\mathbf{x}), \phi_j(\mathbf{x}') \rangle$ for $1 \leq j \leq k$. To efficiently optimize the above optimization problem, we translate it into an equivalent semi-linear infinite program (SILP). The idea behind this transformation is as follows: Let $\Omega(\vec{\alpha}, \vec{\beta})$ be the objective function in Equation (5) and suppose $\vec{\alpha}^*$ is chosen optimally. Then it holds $\Omega(\vec{\alpha}^*, \vec{\beta}) \geq \Omega(\vec{\alpha}, \vec{\beta})$ for all $\vec{\alpha}$ and $\vec{\beta}$. Hence we can equivalently minimize an upper bound θ on the optimal value and by doing so we arrive at the final Optimization Problem 2.

Optimization Problem 2 (SILP). *Given n instances $\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathcal{X}$, either k feature mappings $\phi_1, \ldots, \phi_k$ or alternatively k kernel functions $K_1, \ldots, K_k : \mathcal{X} \times \mathcal{X} \to \Re$ with $K_j(\mathbf{x}, \mathbf{x}') = \langle \phi_j(\mathbf{x}), \phi_j(\mathbf{x}') \rangle$, and $\eta > 0$. The SILP formulation of the feature selection SVDD is given by*

$$\begin{aligned}
\min_{\vec{\beta}, \theta} \quad & \theta \\
s.t. \quad & \theta \geq \sum_{j=1}^{k} \beta_j \left(\sum_{i=1}^{n} \alpha_i K_j(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i,\ell=1}^{n} \alpha_i \alpha_\ell K_j(\mathbf{x}_i, \mathbf{x}_\ell) \right) \\
& \forall \, \vec{\alpha} \in \Re^n : \quad 0 \leq \alpha_i \leq \eta \,, \quad \sum_{i=1}^{n} \alpha_i = 1; \\
& \forall_{j=1}^{k} : \ \beta_j \geq 0 \,; \quad \sum_{j=1}^{k} \beta_j = 1.
\end{aligned}$$

Optimization Problem 2 is equivalent to the primal Optimization Problem 1 and can be efficiently optimized by standard techniques [12].

4. EMPIRICAL EVALUATION

In this section we empirically evaluate the proposed feature selection SVDD on real HTTP network traffic recorded at

```
GET /openworx.php?key=malware+behavior HTTP/1.1\r\n
Host: www.first.fraunhofer.de\r\n
Connection: keep-alive\r\n
Keep-alive: 300\r\n
User-Agent: Mozilla/5.0 (Windows; Windows NT 5.1;
 en-US) Gecko/20070312 Firefox/1.5.0.11\r\n
Cookie: owx_ecrm_keks=b604613a489d40\r\n
Referer: http://www.first.fraunhofer.de/ida\r\n
Accept: image/png,*/*;q=0.5\r\n
Accept-Language: en-us,en;q=0.5\r\n
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7\r\n
Accept-Encoding: gzip,deflate\r\n
```

Figure 1: An exemplary HTTP request.

the Fraunhofer FIRST institute. The unsanitized dataset contains a sample of 2500 normal HTTP requests drawn randomly from two months of incoming HTTP traffic. We injected 30 instances of 10 different attacks taken from recent exploits in the Metasploit framework[1] and a Nessus HTTP scan. All exploits (6 buffer overflow attacks and 4 PHP vulnerabilities) were normalized to match the tokens and frequent attributes of normal HTTP requests such that the malicous payload provides the only indicator for identifying the attacks.

4.1 Feature Extraction

We consider six different feature sets extracted from the raw data. Three of these feature sets are based on a *sequential representation* of byte streams comprising HTTP requests as depicted in Figure 1. The remaining three feature sets correspond to a *token representation* of the HTTP request. The latter is obtained by running requests through an HTTP protocol analyzer constructed with *binpac* [9], and collecting the analysis results in a token-attribute sequence. The tokens in this sequence correspond to keywords of the HTTP protocol whereas the attributes consist of byte sequences associated with these keywords. Figure 2 visualizes the corresponding token-attribute structure of the request in Figure 1. For each of the two representations, we extract the following features from the HTTP requests:

3-gram occurrence features
The feature functions ϕ^{seq}_{occ} and ϕ^{tok}_{occ} register the occurrence of particular byte 3-grams for the sequential and the token representation, respectively. Each feature function is a binary vector where the elements equal 1 if a certain 3-gram occurs in a sequence and 0 otherwise. For sequential representations, this measure is evaluated for the complete byte sequence of the requests. For the token representation, the measure is evaluated separately for all attributes of matching tokens and added up for all tokens.

3-gram frequency features
The computation of the frequency feature functions ϕ^{seq}_{freq} and ϕ^{tok}_{freq} is analogous to the 3-gram occurrence features. The only difference is that both vectors now contain the frequencies of the occurring 3-grams.

Expert features
The feature functions ϕ^{seq}_{exp} and ϕ^{tok}_{seq} exploit the expert knowledge about observed requests. We have chosen a somewhat

[1]http://www.metasploit.com/

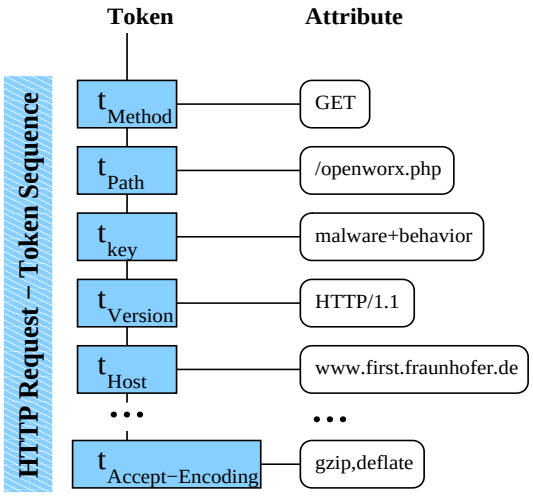


Figure 2: A protocol analyzer returns a set of token-attribute pairs for each HTTP request.

eccentric set of features to show that even a wild guess may be well-suited for the automatic feature selection approach. Our feature set contains 16 features defined as follows. Positions 1 to 11 represent a coarse string length histogram. The range of observed string lengths up to l_{max}, the largest string length in a training corpus, is divided in 10 equally spaced bins. A binary feature is set, if the observed string length falls into the respective bin. Position 11 is reserved for strings exceeding the maximal training string length; this position is always 0 for the training data but may be set to one for the test data. Position 12 is set to one if the entropy of a string lies in the interval $[4.5, 5.5]$. Positions 13–15 flag the occurrence of the following character types in HTTP requests:

- non-printable characters: ANSI numbers 127-255,
- control characters: ASCII numbers 0-31 except for 10 and 13, and
- uncommon characters: \$, [,], {, }, |, \.

Position 16 is set if blacklisted words that are not supposed to appear in a request – in our case: *exec*, *home*, *passthru*, *root*, *CMD* and *SYSTEM* – are found in a string. The difference between sequential and token representations is the same as for the other feature sets.

4.2 Results

We compare the accuracy of the detector obtained by automatic feature selection using the proposed approach with the accuracy of individual detectors using each of the six features separately and a uniform mixture of the features. The respective optimization problems are solved with CPLEX. For the experiments, we randomly draw distinct training, validation, and test sets from the normal pool. The validation and test sets are each augmented by 15 randomly drawn attacks, where we make sure that attacks of the same class occur only in one of the two sets. For every $\eta \in [0, 250]$, each model in our discourse area is adapted to the training set and subsequently tested on the validation set for model selection. Models realizing the largest area under the ROC curve in the false-positive interval $[0, 0.01]$ ($\text{AUC}_{0.01}$) on the

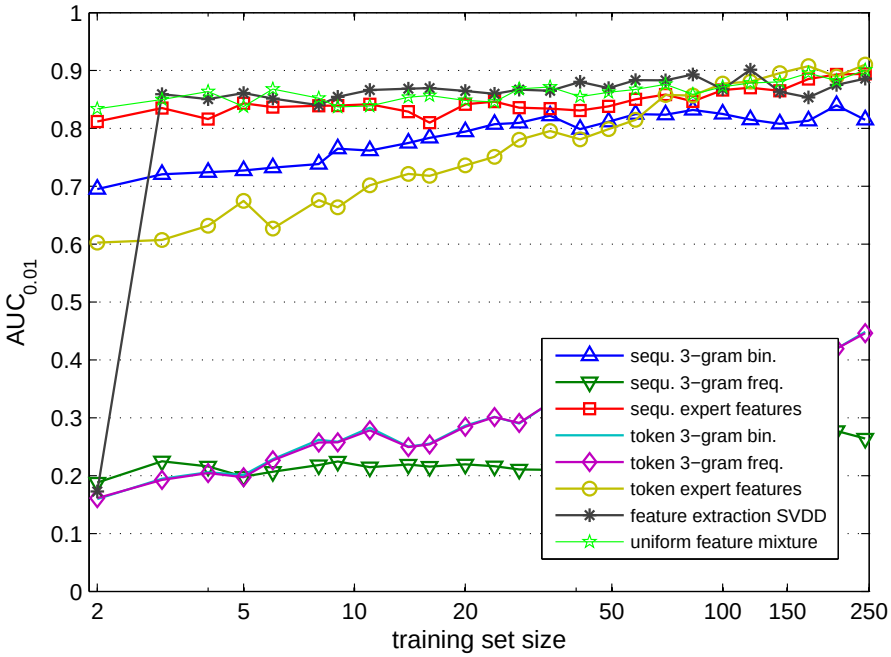


Figure 3: Average $\mathrm{AUC}_{0.01}$ performances for varying training set size.

validation set are then evaluated on the independent test set.

We investigate the accuracy of learned detectors as a function of the training set size. The average $\mathrm{AUC}_{0.01}$ values are reported in Figure 3 over $n = 100$ repetitions with randomly drawn training, validation, and test sets. The standard error (standard deviation divided by $\sqrt{n} = 10$) was observed to be less than 0.01 in our experiments and is not shown in the plots.

It can be clearly seen from Figure 3 that the accuracy of the detector with automatic feature selection dominates the accuracy of all individual classifiers for all training data set sizes (except the ridiculously small training set of size 2). Towards larger training set sizes, some of the features yield equally accurate detectors; the detector obtained by the proposed method remains among the winners.

The behavior of the automatic feature selection becomes clear from the analysis of the distribution of the mixture coefficients for different features shown in Figure 4. Recall that by definition of our problem these features add up to one. It can be seen that for smaller training set sizes, an optimal feature selection is non-sparse, i.e. more than one feature is needed for the best classification. This explains why a strict improvement of the detection accuracy is attained by our method. For larger training sets, the information contained in the data alone becomes sufficient to determine a "strict winner" among the features: in our case, the feature set ϕ^{seq}_{exp}. Although some other feature sets also exhibit good performance for these training set sizes, the choice is made for the feature set with the best overall performance. As a sanity check, we have repeated the experiment with the best feature set replaced by random features and have observed that the best alternative set of features is chosen by automatic feature selection (results not shown in the plots).

5. CONCLUSION

We have presented a novel generalization of the support vector data description (SVDD) that automatically selects the optimal feature combination. The optimization problem of the feature selection SVDD can be formulated as a semi-infinite linear program and solved with standard techniques. The vanilla SVDD is obtained as a special case for only a single feature function. Empirically, the automatic feature selection proved robust against noise in the training data: Fluctuations caused by small sample sizes are absorbed by appropriately chosen mixtures. The feature selection SVDD has consistently outperformed any baseline using only a single feature set.

The proposed method for feature selection for anomaly detection shows that multiple features sets, possibly resulting from various characterizations of the normal traffic, can be *automatically combined* to obtain optimal detectors. In this way a practitioner faced with the choice of alternative feature sets need not make an a priori choice by hand but can rely on the same learning algorithm used to derive the model of normal data.

The future work will focus on optimizing the run-time of the proposed method (currently our implementation uses standard optimization software not suitable to more than a few hundred examples, however for other types of machine learning these kinds of methods have been shown to scale to thousands of training examples [1, 12, 17]), as well as to extend the proposed method to other anomaly detection algorithms.

Acknowledgements

The authors wishes to thank Konrad Rieck and Alexander Zien for fruitful discussion and helpful comments. This work was supported in part by the German Bundesministerium für Bildung und Forschungt (BMBF) under the project RE-

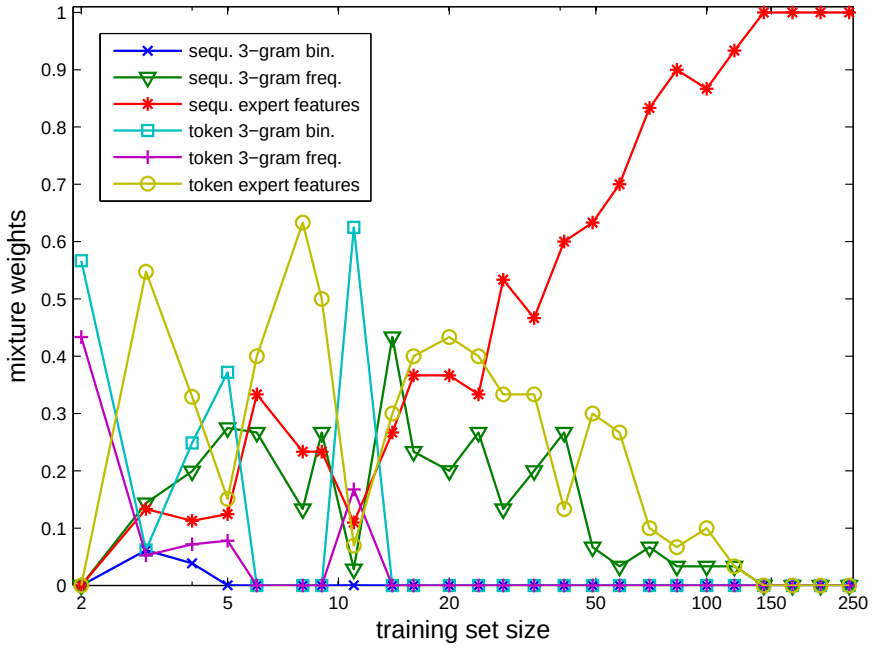


Figure 4: Averaged mixture coefficients $\vec{\beta}$ for varying training set size.

MIND (FKZ 01-IS07007A) and by the FP7-ICT Programme of the European Community, under the PASCAL2 Network of Excellence, ICT-216886.

6. REFERENCES

[1] F. Bach, G. Lanckriet, and M. Jordan. Multiple kernel learning, conic duality, and the SMO algorithm. In *Proceedings of the Twenty-first International Conference on Machine Learning*, 2004.

[2] S. Forrest, S. Hofmeyr, A. Somayaji, and T. Longstaff. A sense of self for unix processes. In *Proc. of IEEE Symposium on Security and Privacy*, pages 120–128, Oakland, CA, USA, 1996.

[3] A. Globerson and N. Tishby. Sufficient dimensionality reduction. *Journal of Machine Learning Research*, 3:1307 – 1331, 2003.

[4] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *JMLR*, 3:1157–1182, 2003.

[5] W. Lee and S. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information Systems Security*, 3:227–261, 2000.

[6] M. Mahoney and P. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 376–385, 2002.

[7] M. Mahoney and P. Chan. Learning rules for anomaly detection of hostile network traffic. In *Proc. of International Conference on Data Mining (ICDM)*, 2003.

[8] H.-N. Nguyen and S.-Y. Ohn. Drfe: Dynamic recursive feature elimination for gene identification based on random forest. In *Proceedings of the International Conference on Neural Information Processing*, 2006.

[9] R. Pang, V. Paxson, R. Sommer, and L. L. Peterson. binpac: a yacc for writing application protocol parsers. In *Proc. of ACM Internet Measurement Conference*, pages 289–300, 2006.

[10] K. Rieck and P. Laskov. Detecting unknown network attacks using language models. In *Detection of Intrusions and Malware, and Vulnerability Assessment, Proc. of 3rd DIMVA Conference*, LNCS, pages 74–90, July 2006.

[11] K. Rieck and P. Laskov. Language models for detection of unknown attacks in network traffic. *Journal in Computer Virology*, 2(4):243–256, 2007.

[12] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large Scale Multiple Kernel Learning. *Journal of Machine Learning Research*, 7:1531–1565, July 2006.

[13] D. M. Tax and R. P. Duin. Support vector data description. *Machine Learning*, 54:45–66, 2004.

[14] K. Wang, J. Parekh, and S. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Recent Adances in Intrusion Detection (RAID)*, pages 226–248, 2006.

[15] K. Wang and S. Stolfo. Anomalous payload-based network intrusion detection. In *Recent Adances in Intrusion Detection (RAID)*, pages 203–222, 2004.

[16] H.-L. Wei and S. A. Billings. Feature subset selection and ranking for data dimensionality reduction. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 29(1):162–166, 2007.

[17] A. Zien and C. S. Ong. Multiclass multiple kernel learning. In Z. Ghahramani, editor, *ICML*, volume 227 of *ACM International Conference Proceeding Series*, pages 1191–1198. ACM, 2007.

4.3 Summary

This contribution presents a novel automated feature selection method for anomaly detection which allows for the selection and combination of an optimal set of discriminative features taken from different feature sets to realize a minimal-volume data description. Experiments showed that the proposed method outperforms or compares to any baseline using a single set of features using for than five samples to train the model of normality. Moreover, experimental results suggest, that the proposed method is particularly effective on small training set sizes. By measuring the detection accuracy over different training set sizes, results suggest that the feature weighting depends on the size of training data. While for smaller sized training set the feature weight distribution tends to be uniform, the weight distribution is increasingly dominated by a specific feature set (i.e. sequential expert features) with growing training set size. By increasing the size of the training set the information contained in the the data alone becomes more and more sufficient for one particular feature set to describe “normal” data.

CHAPTER 5

Incorporation of Application Layer Protocol Syntax into Anomaly Detection

5.1 Introduction

The vast majority of today's cyber attacks is carried at the application layer exploiting vulnerabilities in the application or third party components delivering untrusted, malicious user input to the application. Hence, the analysis of application layer content in network traffic has become increasingly important. As a continuative study to Chapter **??**, Chapter 5 introduces a SVM-based anomaly detection method that allows for the detection of unknown network attacks based on syntax-sequential features extracted from parsed network packet payloads containing HTTP requests. To this end, this section introduces a novel composite similarity which is used to calculate pairwise similarity of HTTP requests based on the sequential similarity of byte-sequence components of matching syntactic tokens. To this end, this contribution conducts an experimental evaluation over HTTP traffic using a One-class SVM to investigate to what extent conventional language models over network packet payloads capture discriminative information sufficient to detect unknown application-level attacks, such as Structured Query Language (SQL) injections or Cross-site Scripting (XSS) attacks, and furthermore evaluate to what extent utilization of network protocol context helps improving detection accuracy. To this end, this contribution provides a novel method that extends payload-based anomaly detection by incorporating network protocol structure represented as abstract syntax trees into anomaly detection. A novel kernel function – the *attributed token kernel* – is proposed which defines distance between two data points based on the pairwise comparison of syntax-tagged substrings of HTTP requests. The novel method is evaluated using full-fledged cross validation and compared against a state-of-the-art service-specific anomaly detection method [45] as well as the signature-based attack detection solution *Snort*. The experimental evaluation considers two scenarios. Under the first scenario, we investigate and compare the detection accuracy of the proposed attributed token kernel against the spectrum kernel on unknown overflow-based attacks simulated against a research institution's website (FIRST07). Under the second scenario, both kernels are evaluated on a different, more complex HTTP dataset (NYT08) containing a set of recorded web application attacks, such as SQL or XSS injections.

5.2 Publication

Incorporation of Application Layer Protocol Syntax into Anomaly Detection

Patrick Düssel[1], Christian Gehl[1], Pavel Laskov[1,2], and Konrad Rieck[1]

[1] Fraunhofer Institute FIRST
Intelligent Data Analysis, Berlin, Germany
[2] University of Tübingen
Wilhelm-Schickard-Institute for Computer Science, Tübingen, Germany
{duessel,gehl,laskov,rieck}@first.fraunhofer.de

Abstract. The syntax of application layer protocols carries valuable information for network intrusion detection. Hence, the majority of modern IDS perform some form of protocol analysis to refine their signatures with application layer context. Protocol analysis, however, has been mainly used for misuse detection, which limits its application for the detection of unknown and novel attacks. In this contribution we address the issue of incorporating application layer context into anomaly-based intrusion detection. We extend a payload-based anomaly detection method by incorporating structural information obtained from a protocol analyzer. The basis for our extension is computation of similarity between attributed tokens derived from a protocol grammar. The enhanced anomaly detection method is evaluated in experiments on detection of web attacks, yielding an improvement of detection accuracy of 49%. While byte-level anomaly detection is sufficient for detection of buffer overflow attacks, identification of recent attacks such as SQL and PHP code injection strongly depends on the availability of application layer context.

Keywords: Anomaly Detection, Protocol Analysis, Web Security.

1 Introduction

Analysis of application layer content of network traffic is getting increasingly important for protecting modern distributed systems against remote attacks. In many cases such systems must deal with untrusted communication parties, e.g. the majority of web-based applications. Application-specific attack can only be detected by monitoring the content of a respective application layer protocol.

Signature-based intrusion detection systems (IDS) possess a number of mechanisms for analyzing the application layer protocol content ranging from simple payload scanning for specific byte patterns, as in Snort [19], to protocol analysis coupled with a specialized language for writing signatures and policy scripts, as in Bro [15]. By understanding the protocol context of potential attack patterns, significant improvements in the detection accuracy of unknown application layer attacks can be achieved.

R. Sekar and A.K. Pujari (Eds.): ICISS 2008, LNCS 5352, pp. 188–202, 2008.

The main drawback of signature-based IDS is their dependence on the availability of appropriate exploit signatures. Rapid development of new exploits and their growing variability make keeping signatures up-to-date a daunting if not impossible task. This motivates investigation of alternative techniques, such as anomaly detection, that are capable to detect previously unknown attacks.

Incorporation of the protocol context into anomaly detection techniques is, however, a difficult task. Unlike a signature-based system which looks for a *specific pattern* within a *specific context*, an anomaly-based system must translate *general knowledge* about patterns and their context into a numeric measure of abnormality. The latter is usually measured by a distance from some typical "profile" of a normal event. Hence, incorporation of protocol syntax into the computation of distances between network events (e.g. packets, TCP connections etc.) is needed in order to give anomaly detection algorithm access to protocol context.

The idea behind the proposed method for syntactically aware comparison of network event is roughly the following. A protocol analyzer can transform a byte stream of each event into a structured representation, e.g. a sequence of token/attribute pairs. The tokens correspond to particular syntactic constructs of a protocol. The attributes are byte sequences attached to the syntactic elements of a protocol. Measuring similarity between sequences is well understood, for general object sequences [18], as well as byte streams of network events [16; 17; 23]. To compare sequences of token/attribute pairs we perform computation of sequential similarity at two levels: for sequences of tokens (with partial ordering) and byte sequence values of corresponding tokens. The resulting similarity measure, which we call the *attributed token kernel*, can be transformed into a Euclidean distance easily handled by most anomaly detection algorithms.

To illustrate effectiveness of the protocol syntax-aware anomaly detection, we apply the proposed method for detection of web application attacks. Such attacks, for example SQL injection, cross-site scripting (XSS) and other script injection attacks, are particularly difficult for detection due to (a) their variability, which makes development of signature a futile exercise, and (b) entanglement of attack vector within the protocol framework, which makes simple byte-level content analysis ineffective. Our experiments carried out on client-side HTTP traffic demonstrate a strong performance improvement for these kinds of attacks compared to byte-level analysis. The proposed method should be easily adaptable for other application layer protocols for which a protocol dissector is available.

2 Related Work

A large amount of previous work in the domain of network intrusion detection systems has focused on features derived from network and transport layer protocols. An example of such features can be found in the data mining approach of Lee and Stolfo [9], containing packet, connection and time window features derived from IP and TCP headers. The same work has pioneered the use of

"content" features that comprised selected application-level properties such as number shell prompts, number of failed login prompts, etc. deemed to be relevant for detection of specific attacks. Similar features comprising selected keywords from application layer protocols have been used by Mahoney and Chan for anomaly detection [12].

General content-based features using payload distribution of specific byte groups have been first proposed by Kruegel et al. [7] in the context of service-specific anomaly detection using separate normality models for different application layer protocols. Full distributions of byte values have been considered by Wang and Stolfo [23], extended to models of various languages that can be defined over byte sequences, e.g. n-grams [16; 22].

Incorporation of application-level protocol information into the detection process has been first realized in signature-based IDS. Robust and efficient protocol parsers have been developed for the Bro IDS [15]; however, until recently they were tightly coupled with Bro's signature engine, which has prevented their use in other systems. The development of a stand-alone protocol parser binpac [14] has provided a possibility for combining protocol parsing with other detection techniques. Especially attractive features of binpac are incremental and bi-directional parsing as well as error recovery. These issues are treated in depth in the recent. Similar properties at a more abstract level are exhibited by the recent interpreted protocol analyzer GAPAL [1].

Combination of protocol parsing and anomaly detection still remains largely unexplored. By considering separate models corresponding to specific URI attributes in the HTTP protocol, Kruegel and Vigna [8] have developed a highly effective system for the detection of web attacks. The system combines models built for specific features, such as length and character distribution, defined for attributes of applications associated with particular URI paths. Ingham et al. [6] learn a generalized DFA representation of tokenized HTTP requests using delimiters defined by the protocol. The DFA inference and the n-grams defined over an alphabet of protocol tokens performed significantly better than other content-based methods in a recent empirical evaluation [5]. Our approach differs from the work of Ingham and Inoue in that our method explicitly operates on a two-tier representation – namely token/attribute pairs – obtained from a full protocol analyzer (binpac/Bro) which provides a more fine-grained view on HTTP traffic.

3 Methodology

A payload-based anomaly detection approach benefits from its ability to cope with unknown attacks. The architecture of our system which is specifically built for the requirements of anomaly detection at application layer is illustrated in Fig. 1. The following four stages outline the essential building blocks of our approach and will be explained in detail for the rest of this section.

1. **Protocol Analysis.** Inbound packets are captured from the network by Bro which provides robust TCP re-assembly and forwards incoming packets

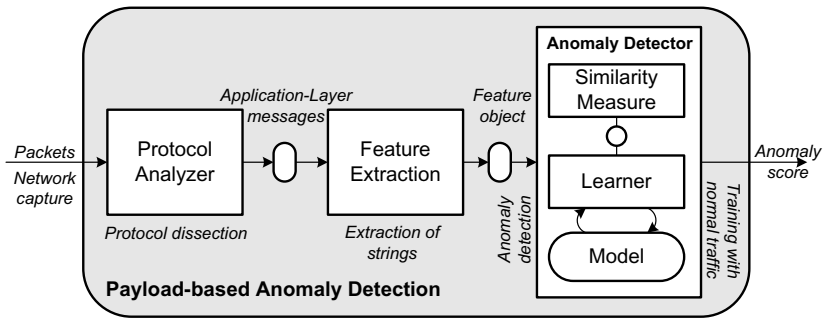


Fig. 1. Architecture of payload-based anomaly detection

to the *binpac* protocol analyzer. The latter extracts application-layer events at different levels of granularity, typical for common text protocols such as HTTP, FTP, SMTP or SIP. Initially, extraction starts at the level of request/response messages and can be further refined to specific protocol elements. A key benefit of using protocol dissectors as part of data preprocessing is the capability to incorporate expert knowledge into the feature extraction process. Details on protocol analysis can be found in Section 3.1.

2. **Feature Extraction.** Each parsed event is mapped into a feature vector which reflects essential characteristics. However, an event can be projected into byte-level or syntax-level feature spaces. Our approach allows to combine both. Details of the feature extraction process can be found in Section 3.2.
3. **Similarity Computation.** The similarity computation between strings is a crucial task for payload-based anomaly detection. Once a message is brought into a corresponding vectorial representation two events can be compared by computing their pairwise distance in a high-dimensional geometric space. We extend the common string similarity measures for token/attribute representations provided by a protocol parser, as explained in Section 3.4.
4. **Anomaly Detection.** In an initial training phase the anomaly detection algorithm learns a global model of "normality" which can be interpreted as a center of mass of a subset of training data. At detection time an incoming message is compared to a previously learned model and based on its distance an anomaly score is computed. The anomaly detection process is described in Section 3.3.

3.1 Protocol Analysis

Network protocols specify rules for syntax, semantics, and synchronization for bidirectional data communication between network endpoints. The protocol syntax is usually defined by an augmented Backus-Naur Form.

Our goal is to analyze network traffic based on the grammatical characteristics of an underlying protocol in order to detect network attacks. We perform the analysis at the granularity of protocol elements present in request/response messages. The HTTP protocol definition is a classical representative of such request/response protocols. Additionally, HTTP is the most frequently used protocol for web applications and so we limit our focus to this particular specification.

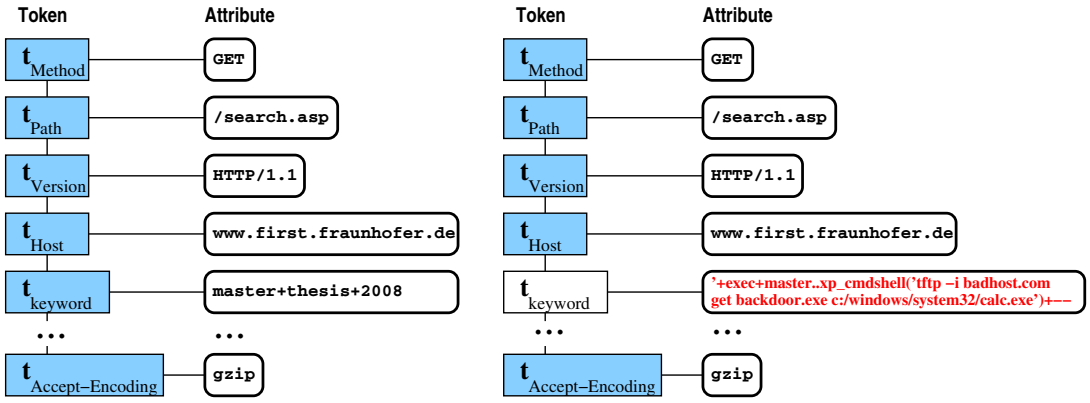


Fig. 2. Attributed token sequence of a benign and malicious HTTP request

Usually, a request is transmitted in a single TCP packet, although certain features of the TCP/IP (e.g. fragmentation) can make the process more complicated. An application protocol analyzer, e.g. binpac [14], allows one to transform the network event's raw byte payload into a structured representation reflecting the syntactic aspects of an underlying application protocol. For example in the syntactic context of HTTP the sequence "Content-Length: 169" refers to a header "Content-Length" with attribute "169".

We present two examples of HTTP connections to illustrate the effect of applying binpac's grammar and parser to an application-level specification. The first example is a benign GET request containing common HTTP headers together with a CGI parameter.

```
GET /search.asp?keyword=master+thesis+learning HTTP/1.1\r\nHost: www.firs
t.fraunhofer.de\r\nUser-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows
 NT 5.1; SV1)\r\nConnection: Keep-alive\r\nAccept-Encoding: gzip\r\n\r\n
```

The second example shows a SQL injection attack against a Microsoft SQL Server exploiting the vulnerable CGI parameter *keyword*.

```
GET /search.asp?keyword='+exec+master..xp_cmdshell('tftp -i badhost.com g
et backdoor.exe c:/windows/system32/calc.exe')+-- HTTP/1.1\r\nHost: www.f
```

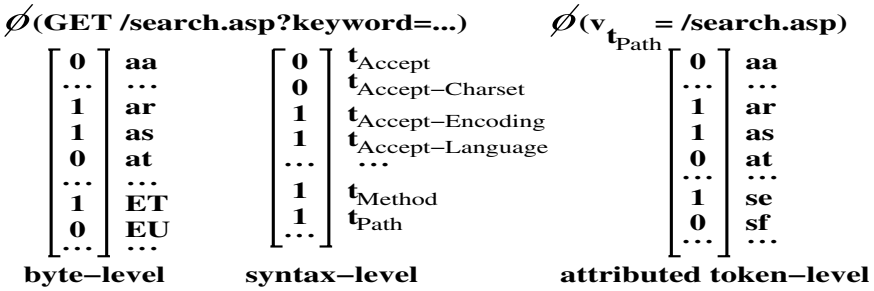


Fig. 3. Feature extraction for byte-level, syntax-level and attributed token-level

```
irst.fraunhofer.de\r\nUser-Agent: Mozilla/5.0\r\nConnection: Keep-alive\r
\nAccept-Language: en-us,en\r\nAccept-Encoding: gzip\r\n\r\n
```

The token/attribute pairs generated by the protocol analyzer are shown in Fig. 2. One can clearly see that the difference between these two requests is given by the attributes attached to the token "keyword" of the parsed GET request.

3.2 Feature Extraction

Anomaly detection usually requires data to be in a vectorial representation. Therefore, the feature extraction process maps application layer messages, such as HTTP requests, into a feature space in which similarity between messages can be computed. Protocol dissection in the pre-processing stage allows to deploy feature extraction on two different levels explained in the following.

Byte-Level Features. An intuitive way to represent a message at byte level is to extract unique substrings by moving a sliding window of a particular length n over a message. The resulting set of feature strings are called n-grams. The first example in Fig. 3 shows the mapping of the benign HTTP request introduced in Section 3.1 into a binary 2-gram feature space.

Syntax-Level Features. Each message is transformed into a set of tokens. Although shown as a sequence of tokens in Fig. 2, these feature have only partial sequential order as the order of many (but not all) tokens in an HTTP request is not defined. An embedding of the benign HTTP request into a token feature space is exemplarily presented in Fig. 3.

With the extraction of byte-level features from token attributes a semantic notion is implicitly assigned to the corresponding protocol token. Note, that an explicit representation of application layer messages can easily become computational infeasible due to the combinatorial nature of byte sequences. Therefore, we use efficient data structures such as suffix trees or hash tables ("feature objects" in Fig. 1) that allow an implicit vectorial representation.

3.3 Anomaly Detection

Once, application layer messages are mapped into some feature space the problem of anomaly detection can be solved mathematically considering the geometric relationship between vectorial representations of messages. Although anomaly detection methods have been successfully applied to different problems in intrusion detection, e.g. identification of anomalous program behavior [e.g. 3; 4], anomalous packet headers [e.g. 11] or anomalous network payloads [e.g. 8; 16; 17; 22; 23], all methods share the same concept – *anomalies are deviations from a model of normality* – and differ in concrete notions of normality and deviation. For our purpose we use the one-class support vector machine (OC-SVM) proposed in [21] which fits a minimal enclosing hypersphere to the data which is characterized by a center c and a radius R as illustrated in Fig. 4.

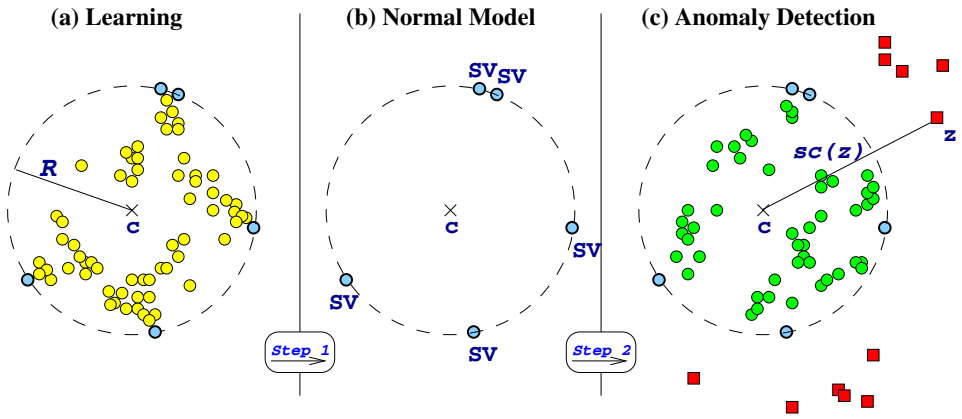


Fig. 4. Learning and anomaly detection using a one-class support vector machine

Mathematically, this can be formulated as a quadratic programming optimization problem:

$$\begin{aligned} \min_{\substack{R\in\mathbb{R}\\ \xi\in\mathbb{R}^n}} \quad & R^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to:} \quad & ||\phi(x_i) - c||^2 \leq R^2 + \xi_i, \\ & \xi_i \geq 0. \end{aligned} \tag{1}$$

By minimizing R^2 the volume of the hypersphere is minimized given the constraint that training objects are still contained in the sphere which can be expressed by the constraint in Eq.(1). A major benefit of this approach is the control of generalization ability of the algorithm [13], which enables one to cope with noise in the training data and thus dispense with laborious sanitization, as recently proposed by Cretu et al. [2]. By introducing slack variables ξ_i and penalizing the cost function we allow the constraint to be softened. The regularization parameter C controls the trade-off between radius and errors (number of training points that violate the constraint). The solution of the optimization problem shown in Eq. (1) yields two important facts:

1. The center $c = \sum_i \alpha_i \phi(x_i)$ of the sphere is a linear combination of data points, while α_i is a sparse vector that determines the contribution of the i-th data point to the center. A small number of training points having $\alpha_i > 0$ are called **support vectors** (SV) which define the model of normality as illustrated in Fig. 4.
2. The radius R which is explicitly given by the solution of the optimization problem in Eq. (1) refers to the distance from the center c of the sphere to the boundary (defined by the set of support vectors) and can be interpreted as a threshold for a decision function.

Finally, having determined a model of normality the anomaly score $sc(z)$ for a test object z can be defined as the distance from the center:

$$sc(z) = ||\phi(z) - c||^2 = k(z,z) - 2\sum_i \alpha_i k(z, x_i) + \sum_{i,j} \alpha_i \alpha_j k(x_i, x_j), \tag{2}$$

where the similarity measure $k(x, y)$ between two points x and y refers to a *kernel* function which allows to compute similarity between two data points embedded in some geometric feature space $\mathcal{F}$. Given an arbitrary mapping $\phi : \mathcal{X} \mapsto \mathcal{F}$ of a data point $x \in \mathcal{X}$ a common similarity measure can be defined from the *dot product* in $\mathcal{F}$:

$$k(x, y) = \langle \phi(x), \phi(y) \rangle = \sum_{i=1}^{N} \phi_i(x)\phi_i(y), \tag{3}$$

where $\phi_i(x)$ refers to the i-th dimension of a data point x mapped into $\mathcal{F}$.

3.4 Similarity Measures

Having defined a set of characteristic features we can develop similarity measures that, given two data points, returns a real number reflecting their similarity.

Spectrum Kernel. A natural way to define a kernel $k(s, u)$ between two application layer messages s and u is to consider n-grams that both messages have in common. Given the set $\mathcal{A}^n$ of all possible strings of length n induced by an alphabet $\mathcal{A}$ we can define a kernel function computing the dot product of both messages embedded into a $|\mathcal{A}|^n$ dimensional feature space.

$$k(s, u) = \langle \phi(s), \phi(u) \rangle = \sum_{w \in \mathcal{A}^n} \phi_w(s)\phi_w(u), \tag{4}$$

where $\phi_w(s)$[1] refers to a signum function that returns 1 if the w is contained in s and 0 otherwise. The kernel function $k(s, u)$ is referred to as *spectrum kernel* [10]. Using n-grams to characterize messages is intuitive but may result in a high-dimensional feature space. From an algorithmic point of view, it may seem that running a summation over all possible substrings $w \in \mathcal{A}^n$ can become computationally infeasible. Thus, special data structures such as tries, suffix trees or suffix arrays enable one to compute $k(s, u)$ in $O(|s| + |u|)$ time [20]. Interestingly, the Euclidean distance $d_{eucl}(s, u)$ which is of particular interest for anomaly detection can be easily derived from the above kernel formulation:

$$d_{eucl}(s, u) = \sqrt{k(s, s) + k(u, u) - 2k(s, u)}. \tag{5}$$

Attributed Token Kernel. In this section we address the problem of how to combine string similarity as defined in Section 3.4 and structural similarity of two application layer messages. Consider an alphabet $\mathcal{A} = t_1, t_2, \ldots, t_z$ of tokens. Let $s = s_1, s_2, ..., s_n$ and $u = u_1, u_2, ..., u_m$, $s_i, u_j \in \mathcal{A}$, be the token sets of two application layer messages returned by a protocol analyzer. Let v_t^u denote an attribute attached to the token t found in a token set u. We can define a kernel function for attributes in the same way we have done it in Eq.(4):

$$k_t(s, u) = \begin{cases} k(v_t^s, v_t^u), & \text{if } t \text{ is found in both } s \text{ and } u \\ 0, & \text{otherwise.} \end{cases} \tag{6}$$

[1] Alternatively, the embedding function $\phi_w(s)$ may return the count or the term frequency of w in s.

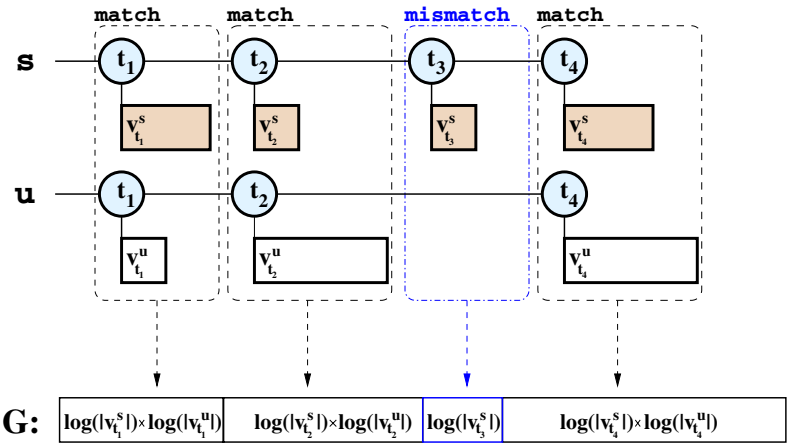


Fig. 5. Normalization of the similarity measure between attributed token sequences

Finally, we combine definitions (4) and (6) in the following way:

$$k(s,u) = \sum_{t \in s \cap u} \frac{\gamma(t)}{G} k_t(s,u), \tag{7}$$

where $s \cap u$ is an intersection of tokens in s and u, $\gamma(t)$ is a weight assigned to a particular token, and G is an overall normalization constant. The computation of the kernel function in Eq. (7) runs through all matching tokens and computes the similarity measure defined in Eq. (4) over associated attributes at byte level.

The weighting constant is defined as:

$$\gamma(t) = log(|v_t^s|) \times log(|v_t^u|),$$

and the normalization constant is defined as:

$$G = \sum_{t \in s \cup u} \gamma(t).$$

These constants are motivated by the need to normalize contributions from individual value sequences according to their lengths. The overall normalization constant also includes contributions from mismatching tokens. This allows the latter to indirectly influence the similarity measure by rescaling contributions from matching tokens; direct influence is not possible since it does not make any sense to compare value strings for mismatching tokens. For the rest of this paper we refer to the kernel function defined in Eq. (7) as *attributed token kernel*.

4 Experiments

We evaluate the impact of incorporation of protocol context into anomaly detection on two data sets containing HTTP traffic. Both data sets comprise exploits taken from the Metasploit framework[2] as well as from common security mailing lists and archives such as xssed.com, sla.ckers.org or Bugtraq.

[2] http://www.metasploit.com/

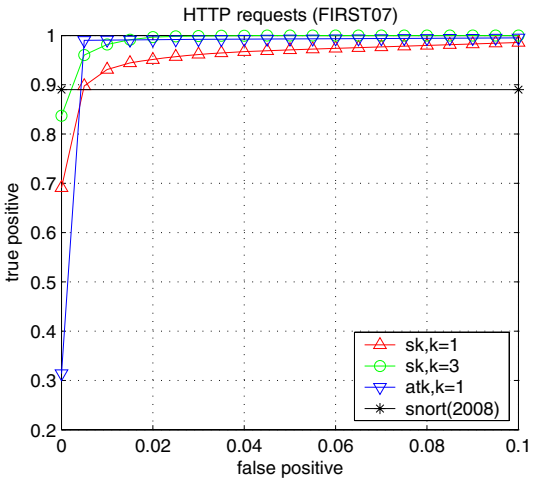
The first dataset (FIRST07) contains a sample of 16000 normal HTTP connections drawn from two months incoming traffic recorded at our institute's web server. We mixed normal data with 42 attack instances of 14 different types, mostly **overflow-based attacks** (8 stack overflows, 4 heap overflows, 1 format string exploit, 1 web application attack). Except for using unsanitized data, this a typical experimental protocol used in previous work, e.g. [5; 8; 23].

The second data set (NYT08) has been motivated by the observation that a traffic profile of a mostly static web site may be quite different from a profile of a site running web applications. In particular, a much more involved structure of URI and certain header fields can be expected in normal traffic, which may cause significantly higher false positive rates than the ones reported in evaluations on static normal traffic. Since we do not have access to a "pure" web application traffic, as e.g. the Google data used by Kruegel and Vigna [8], we have attempted to simulate such traffic using specially developed crawlers. Our request engine analyzes the structure and content of existing static traffic (in our case, the FIRST07 data) and generates valid HTTP requests using frequently observed protocol header elements while randomly visiting a target domain. We generated 15000 client-side connections accessing the *nytimes.com* news site and mixed the traffic with 17 instances of **web application attacks** (1 buffer overflow, 1 arbitrary command execution vulnerability, 3 Perl injection, 2 PHP include, 4 SQL injections and 6 XSS attacks). XSS attacks and SQL injections were launched against two prototypical CGI programs that were adapted to the structure of a "login"-site (8 CGI parameters) and a "search"-site (18 CGI parameters) found in the *nytimes.com* domain. In order to obtain a normal behavior for these sites we generated 1000 requests containing varying user names and passwords as well as search phrases and realistic parameter values for both prototypical "mirrors" of NYT sites. In contrast to previous work in which the structure of HTTP requests in exploits was used "as is", we have also normalized the attacks by using the same typical headers injected by crawlers, in order to avoid attacks being flagged as anomalous purely because of programmatic but non-essential difference in their request structure.

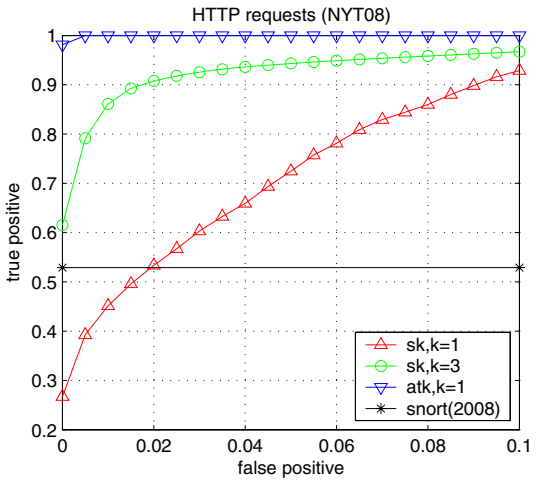
In our experiments, a model was trained using 500 requests taken from a normal pool of data and subsequently applied to 500 unknown requests taken from a distinct test set mixed with attacks. The detection accuracy was measured in terms of area under receiver operating characteristic curve ($ROC_{0.1}$) for false positive rates $\leq 10\%$. For statistical reasons experiments were repeated and the detection accuracy was averaged over 50 repetitions.

4.1 Detection Accuracy: Byte-Level Versus Attributed Token-Level

In the experiment on FIRST07 we investigate the detection of overflow-based attacks in HTTP requests. As shown in Fig. 6(a), the spectrum kernel **sk** on binary 3-grams attains a detection rate of 82% at 0% false positives anomaly, which is comparable to Snort. The only attack that repeatedly suffered a high false positive rate (1.5%-2%) is a *file inclusion* ("php_include") which is the only non-buffer-overflow attack in this data set. Similar results have been obtained



(a) ROC curve of OC-SVM on the FIRST07 data set (overflow attacks)



(b) ROC curve of OC-SVM on the NYT08 data set (web application attacks)

Fig. 6. Detection accuracy on intrusion detection datasets FIRST07 and NYT08

for the attributed token kernel except for some initial false positives due to proxy requests containing anonymized header attributes.

In the experiment on NYT08 we investigate the detection of attacks that are bound to specific parameters of a CGI program. The results presented in Fig. 6(b) reveal that the detection of *web application attacks* using byte-level similarity over n-grams is much more difficult than for overflow-based attacks (Fig. 6(a)). The OC-SVM achieves its best detection rate of approximately 64% at zero percent false positives using a spectrum kernel (**sk**) over 3-grams and a binary feature embedding. Moreover, it can be observed that the accuracy of byte-level detection rises by increasing the size of n-grams. As illustrated in Fig. 6(b) significant improvements can be obtained by incorporating structure of application layer messages into payload-based anomaly detection as illustrated in Fig. 2. It turns out that anomaly detection using the attributed token kernel (**atk**) achieves a 97% detection rate without suffering any false positive.

In order to assess the computational effort of our method we measured the runtime for involved processing steps. Parsing and feature extraction require on average 1.2ms per request; average kernel computation and anomaly detection take 3.5ms per request. The overall processing time of 4.7ms per request yields a throughput of approximately 212 HTTP requests per second.

4.2 Visualization of Discriminative Features

In order to illustrate the increase of discriminative power gained through incorporation of application layer context into anomaly detection Fig. 4.2 displays 1-gram frequency differences between 1000 normal HTTP requests and two different attacks, a buffer overflow and a SQL injection. A frequency difference of 1 arises from bytes that only appear in normal data points whereas bytes that can be exclusively observed in the attack instance result in a frequency difference of -1. Bytes with a frequency difference close to zero do not provide

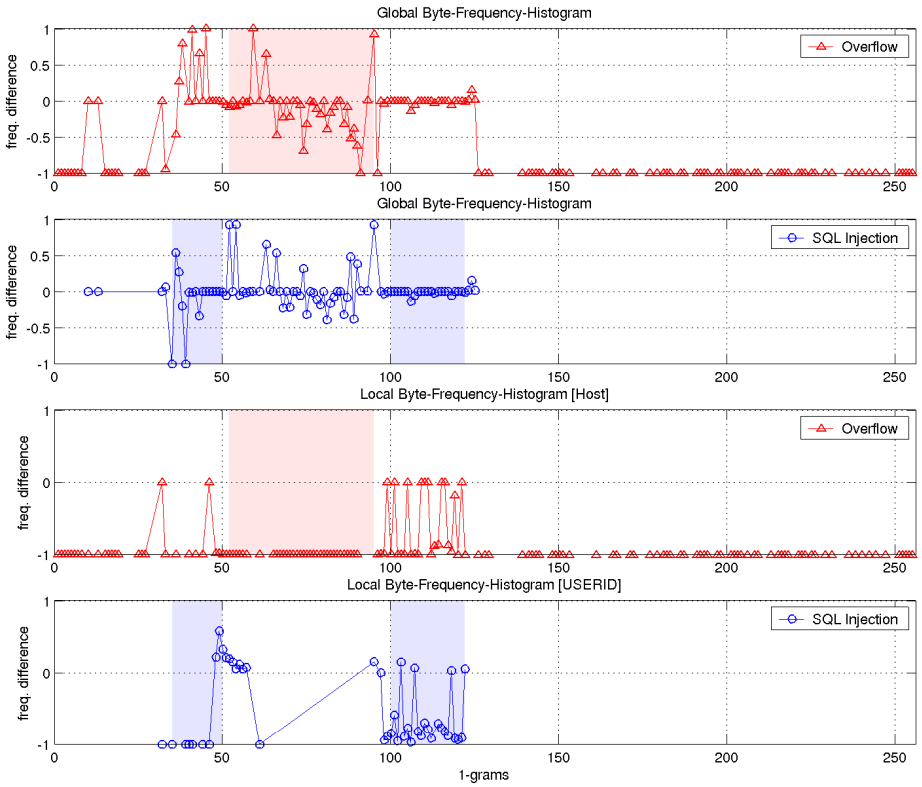


Fig. 7. Byte-frequency differences at request-level and token-level

discriminative information. The upper two charts display frequency differences for a buffer overflow attack ("edirectory_host_shikata_ga_nai") and a SQL injection ("sql_union_injection") that results from conventional byte-level analysis. The buffer overflow can be easily detected due to the presence of a large amount of non-printable characters (a large number of points at -1 for lower byte values and values above 128). On the other hand, the SQL injection contains mostly ASCII characters, which is reflected by close to zero frequency differences for most of the printable characters. As a consequence, the detection of this attack is relatively difficult. The lower two charts show the frequency differences for the same attacks within their appropriate application layer context. The frequency differences of the buffer overflow attack become even more obvious by examining the local byte distribution within the exploited request parameter "Host". Similar clarification takes place for the SQL injection attack for which many previously normal bytes become clearly anomalous considering the CGI parameter "USERID". This results in a major improvement of detection accuracy.

4.3 Comparison with Other Methods

To give a comparison to different intrusion detection techniques we examined a model-based anomaly detection method (**model-based AD**) proposed by Kruegel and Vigna [8] and the well-known signature IDS **Snort**[3]. The model-based detection method applies a number of different models to individual

[3] VRT Certified Rules (registered user release) downloaded 07/15/2008.

Table 1. Comparison of fp-rates between model-based AD, Snort and OC-SVM

HTTP attacks (NYT08 dataset)	model-based AD			Snort	OC-SVM	
	ALM	ICD	Combined		sk	atk
edirectory_host_alpha_mixed	.0728	.0208	.0362	+	.0	.0
sql_1=1	.0112	.0379	.0214	−	.1798	.0002
sql_backdoor	.0	.0006	.0	+	.0004	.0
sql_fileloader	.0025	.0006	.0010	−	.0065	.0
sql_union_injection	.0	.0020	.0	−	.0047	.0
xss_alert	.0	.0	.0	+	.0388	.0
xss_dom_injection	.0	.0	.0	+	.0001	.0
xss_img_injection	.0	.0	.0	−	.0004	.0

parameters of HTTP queries. By learning various parameter models this method creates a "user profile" for each server-side program that is compared against incoming requests for a particular resource. To allow a fair comparison to our method we build models of not only CGI parameters in the URI, but also of header parameters and CGI parameters present in the request's body.

In our experiments the model-based AD comprises two models:

Attribute length model (ALM) estimates the attribute length distribution of CGI parameters and detects data points that significantly deviate from normal models.

Attribute character distribution model learns the *idealized character distribution* (ICD) of a given attribute. Using ICD instances are detected whose sorted frequencies differ from the previously learned frequency profile.

A comparison of false positive rates per attack class (percentage of false positives in test set given a detection of all instances from that attack class) is provided in Table 1. Anomaly detection using the attributed token kernel exhibits almost perfect accuracy of the 8 selected attacks, whereas both models of Kruegel and Vigna as well as their combination suffer from significant false positive rates for the first two attacks. Interestingly, Snort reported alarms for most of the cross site scripting and buffer overflow instances but failed to detect the majority of SQL injections which shows that specific exploits may require customization of signatures in order to provide a consistent protection.

5 Conclusions

In this paper, we have developed a general method for the incorporation of application layer protocol syntax into anomaly detection. The key instrument of our method is computation of similarity between token/attribute sequences that can be obtained from a protocol analyzer. A combined similarity measure is developed for such sequences which takes into account the syntactic context contained in tokens as well as the byte-level payload semantics.

The proposed method has proved to be especially useful for the detection of web application attacks. We have carried out experiments on realistic traffic

using the HTTP protocol analyzer developed in binpac. Although the additional effort of protocol analysis does not pay off for simple buffer overflow exploits, the detection rate for web application attacks has been boosted from 70% to 100% in the false positive rate interval of less than 0.14%. Surprisingly, our method has even outperformed a very effective model-based method of Kruegel and Vigna [8] that was specially designed for the detection of web application attacks.

Due to its generality, the proposed method can be used with any other application layer protocol for which a protocol analyzer is available. It can be deployed in a variety of distributed systems applications, especially the ones for which very few examples of potential exploits are currently known (e.g. IP multimedia infrastructure and SCADA systems).

Acknowledgements. This work was supported by the German Bundesministerium für Bildung und Forschung (BMBF) under the project ReMIND (FKZ 01-IS07007A).

References

[1] Borisov, N., Brumley, D., Wang, H., Dunagan, J., Joshi, P., Guo, C.: Generic application-level protocol analyzer and its language. In: Proc. of Network and Distributed System Security Symposium (NDSS) (2007)

[2] Cretu, G., Stavrou, A., Locasto, M., Stolfo, S., Keromytis, A.: Casting out demons: Sanitizing training data for anomaly sensors. In: ieeesp (to appear, 2008)

[3] Forrest, S., Hofmeyr, S., Somayaji, A., Longstaff, T.: A sense of self for unix processes. In: Proc. of IEEE Symposium on Security and Privacy, Oakland, CA, USA, pp. 120–128 (1996)

[4] Gao, D., Reiter, M., Song, D.: Behavioral distance measurement using hidden markov models. In: Zamboni, D., Krügel, C. (eds.) RAID 2006. LNCS, vol. 4219, pp. 19–40. Springer, Heidelberg (2006)

[5] Ingham, K.L., Inoue, H.: Comparing anomaly detection techniques for http. In: Kruegel, C., Lippmann, R., Clark, A. (eds.) RAID 2007. LNCS, vol. 4637, pp. 42–62. Springer, Heidelberg (2007)

[6] Ingham, K.L., Somayaji, A., Burge, J., Forrest, S.: Learning dfa representations of http for protecting web applications. Computer Networks 51(5), 1239–1255 (2007)

[7] Kruegel, C., Toth, T., Kirda, E.: Service specific anomaly detection for network intrusion detection. In: Proc. of ACM Symposium on Applied Computing, pp. 201–208 (2002)

[8] Kruegel, C., Vigna, G.: Anomaly detection of web-based attacks. In: Proc. of 10th ACM Conf. on Computer and Communications Security, pp. 251–261 (2003)

[9] Lee, W., Stolfo, S.: A framework for constructing features and models for intrusion detection systems. ACM Transactions on Information Systems Security 3, 227–261 (2000)

[10] Leslie, C., Eskin, E., Noble, W.: The spectrum kernel: A string kernel for SVM protein classification. In: Proc. Pacific Symp. Biocomputing, pp. 564–575 (2002)

[11] Mahoney, M., Chan, P.: PHAD: Packet header anomaly detection for identifying hostile network traffic. Technical Report CS-2001-2, Florida Institute of Technology (2001)

[12] Mahoney, M., Chan, P.: Learning nonstationary models of normal network traffic for detecting novel attacks. In: Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD), pp. 376–385 (2002)
[13] Müller, K.-R., Mika, S., Rätsch, G., Tsuda, K., Schölkopf, B.: An introduction to kernel-based learning algorithms. IEEE Neural Networks 12(2), 181–201 (2001)
[14] Pang, R., Paxson, V., Sommer, R., Peterson, L.: binpac: a yacc for writing application protocol parsers. In: Proc. of ACM Internet Measurement Conference, pp. 289–300 (2006)
[15] Paxson, V.: Bro: a system for detecting network intruders in real-time. In: Proc. of USENIX Security Symposium, pp. 31–51 (1998)
[16] Rieck, K., Laskov, P.: Detecting unknown network attacks using language models. In: Büschkes, R., Laskov, P. (eds.) DIMVA 2006. LNCS, vol. 4064, pp. 74–90. Springer, Heidelberg (2006)
[17] Rieck, K., Laskov, P.: Language models for detection of unknown attacks in network traffic. Journal in Computer Virology 2(4), 243–256 (2007)
[18] Rieck, K., Laskov, P.: Linear-time computation of similarity measures for sequential data. Journal of Machine Learning Research 9, 23–48 (2008)
[19] Roesch, M.: Snort: Lightweight intrusion detection for networks. In: Proc. of USENIX Large Installation System Administration Conference LISA, pp. 229–238 (1999)
[20] Shawe-Taylor, J., Cristianini, N.: Kernel methods for pattern analysis. Cambridge University Press, Cambridge (2004)
[21] Tax, D., Duin, R.: Data domain description by support vectors. In: Verleysen, M. (ed.) Proc. ESANN, Brussels, pp. 251–256. D. Facto Press (1999)
[22] Wang, K., Parekh, J., Stolfo, S.: Anagram: A content anomaly detector resistant to mimicry attack. In: Zamboni, D., Krügel, C. (eds.) RAID 2006. LNCS, vol. 4219, pp. 226–248. Springer, Heidelberg (2006)
[23] Wang, K., Stolfo, S.: Anomalous payload-based network intrusion detection. In: Jonsson, E., Valdes, A., Almgren, M. (eds.) RAID 2004. LNCS, vol. 3224, pp. 203–222. Springer, Heidelberg (2004)

5.3 Summary

This contribution presentsa novel method that incorporates application layer protocol syntax into anomaly detection. To this end, a novel kernel function was defined which allows to calculate pairwise similarity between token-attribute sequences obtained from HTTP requests parsed by the binpac protocol analyzer. Experimental results suggest, that under the first scenario the results for the attributed token kernel compare to the best spectrum kernel (k=1, binary feature embedding) with a 99% true positive rate at less than 1% false positives. This can be explained by the fact, that overflow-based attacks are not difficult to detect, generally due to the inclusion of large network packet payloads and vast utilization of non-printable characters. Under the second scenario, the experimental evaluation shows that attributed token kernel significantly outperforms the spectrum kernel and boosts the detection rate on web application attacks from 70% to 100% at less than 0.14% false positives. Surprisingly, the proposed method also outperforms both misuse-based detection as well as the service-specific anomaly detection approach.

CHAPTER 6

Detecting zero-day attacks using context-aware anomaly detection at the application layer

6.1 Introduction

As the majority of network attacks are carried out at the application level, the analysis of application-layer content in network traffic is getting increasingly important. Payload-based anomaly detection is capable to detect unknown application-level attacks. Thereby, the choice of data representation required to measure pairwise similarity strongly affects the detection accuracy of an anomaly detector. Sequential data representations, such as n-grams over network packet payload bytes [e.g. 46, 47, 91] demonstrate superior accuracy at the detection of unknown overflow-based attacks. However, this type of data representation does not adequately account for structural sensitivity needed to detect inconspicuous types of application-level attacks, such as SQL injections or cross-site scripting. Earlier work has proved the usefulness of incorporating protocol analysis into anomaly detection [e.g. 71]. To this end, this contribution introduces a novel data structure for attributed language models, c_k-grams, which extends earlier work by combining protocol syntax features and sequential features efficiently into one unified feature space and thus, reduce complexity of pairwise similarity calculation as introduced in [e.g. 71] while increasing local outlier sensitivity.

6.2 Publication

International Journal of Information Security manuscript No.
(will be inserted by the editor)

Detecting Zero-Day Attacks Using Context-Aware Anomaly Detection At Application-Layer

Patrick Duessel · Christian Gehl · Ulrich Flegel · Sven Dietrich · Michael Meier

Received: date / Accepted: date

Abstract Anomaly detection allows for the identification of unknown and novel attacks in network traffic. However, current approaches for anomaly detection of network packet payloads are limited to the analysis of plain byte sequences. Experiments have shown that application-layer attacks become difficult to detect in the presence of attack obfuscation using payload customization. The ability to incorporate syntactic context into anomaly detection provides valuable information and increases detection accuracy. In this contribution, we address the issue of incorporating protocol context into payload-based anomaly detection. We present a new data representation, called c_n-grams, that allows to integrate syntactic and sequential features of payloads in an unified feature space and provides the basis for context-aware detection of network intrusions. We conduct experiments on both text-based and binary application-layer protocols which demonstrate superior accuracy on the detection of various types of attacks over regular anomaly detection methods. Furthermore, we show how c_n-grams can be used to interpret detected anomalies and thus, provide explainable decisions in practice.

Keywords Intrusion detection · machine learning · anomaly detection · protocol analysis · deep packet inspection

Patrick Duessel
Deloitte & Touche LLP
Cyber Risk Services | Risk Analytics
30 Rockefeller Plaza, New York, NY 10112-0015, United States
Tel.: +1 (212) 492 2332
E-mail: paduessel@deloitte.com

Christian Gehl
Trifense GmbH - Intelligent Network Defense
Germendorfer Strasse 79, 16727 Velten, Germany
Tel.: + 49 (0) 3304 360 368
E-mail: christian.gehl@trifense.de

Ulrich Flegel
Infineon Technologies AG
Am Campeon 1-12, 86579 Neubiberg, Germany
Tel.: + 49 (0) 89 234 21728
E-mail: ulrich.flegel@infineon.com

Sven Dietrich
CUNY John Jay College of Criminal Justice
Mathematics and Computer Science Department
524 W 59th St, 10019 New York, United States
E-mail: spock@ieee.org

Michael Meier
University of Bonn
Friedrich-Ebert-Allee 144, 5313 Bonn, Germany
Tel.: +49 228 73 54249
E-mail: mm@cs.uni-bonn.de

1 Introduction

The analysis of application-layer content in network traffic is getting increasingly important for the protection of complex business environments which deploy a variety of application-specific services and allow for access and transfer of sensitive data between untrusted communication parties. Nowadays, the majority of attacks is carried out at the application-layer. Therefore, monitoring the content of a respective application-layer protocol becomes vital for the detection of unknown and novel application-specific attacks, so called Zero-day attacks.
Signature-based intrusion detection systems (IDS) possess a number of mechanisms for analyzing application-layer protocol content ranging from byte pattern matching as provided by Snort [28] to sophisticated protocol analysis as realized in Bro [22, 23].

However, the major drawback of signature-based IDS is their reliance on appropriate exploit signatures in order to provide adequate protection. Unfortunately, keeping signatures up-to-date is a tedious and resource intensive task given the rapid development of new exploits and their growing variability. This motivates the investigation of alternative techniques.

Payload-based anomaly detection is capable to instantaneously detect previously unknown application-layer attacks. Unlike signature-based systems which search for *explicit* byte patterns, payload-based anomaly detection systems must translate *general knowledge* about patterns into a numeric measure of abnormality which is usually defined by a distance from some model learned over normal payloads.

Thereby, the choice of data representation which is required to measure similarity between sequential data strongly affects the capabilities of an anomaly detector at hand. Sequential data representations, such as n-grams of payload bytes [25, 34], exhibit superior precision at the detection of unknown overflow-based attacks. However, this type of data representation does not adequately account for structural sensitivity needed for detection of rather inconspicuous looking attacks such as cross-site scripting (XSS) or SQL injection. By accessing protocol context of attack patterns significant improvements in the detection accuracy of unknown application-layer attacks can be achieved [5, 9, 14].

In this article, we propose a novel representation of network payloads that integrates protocol context and byte sequences into an unified feature space and thus, allows for a context-aware detection of network intrusions. To this end, a protocol analyzer transforms a network byte stream into the structured representation of a parse tree. Tree nodes are extracted and inserted as tuples of token/attributes into the c_n-gram data structure, a novel data representation of network traffic that allows to efficiently combine sequential models with protocol tokens. Moreover, explainability and the capability to visualize suspicious content with respect to protocol context is another advantage of this data structure over sequential representations.

In order to illustrate the effectiveness of the proposed context-aware payload analysis, we conduct an extensive experimental evaluation in which c_n-grams are compared to conventional n-grams in the presence of a diversity of attacks carrying various payloads. In order to point out the strengths of the proposed data representation, attacks not only include buffer overflows but also web application attacks. Such attacks, for example SQL injections, XSS injections and other script injection attacks, are particularly difficult to detect due to their variability and their strong entanglement in the protocol framework, which makes content analysis based on sequential features ineffective. Our experiments are carried out on network traffic containing text-based and binary application-layer protocols.

The paper is structured as follows: The main contribution of the paper is presented in Section 2 and provides details on a novel data representation for network payloads which allows for the computation of context-aware sequential similarity by geometric anomaly detection methods. A comprehensive experimental evaluation of the proposed method on network traffic featuring various application-layer protocols is carried out in Section 3. Related work on content-based anomaly detection and protocol analysis is presented in Section 4. Finally, conclusions and an outline of future work can be found in Section 5.

2 Methodology

The following four stages outline the essential building blocks of our approach and will be explained in detail for the rest of this section.

1. **Data Acquisition and Normalization**. Inbound packets are captured from the network, reassembled and forwarded to a protocol analyzer such as *binpac* [21] which allows to extract application-layer messages from both text-based and binary protocols. A key benefit of using protocol dissectors as part of data pre-processing is the capability to incorporate expert knowledge in the subsequent feature extraction process. Details on protocol analysis can be found in Section 2.1.
2. **Feature Extraction**. At this stage, byte messages are mapped into a metric space using data representations and features which reflect essential characteristics of a byte sequence. Our approach allows to combine byte-level and syntax-level features in an unified metric space. Details of the feature extraction process can be found in Section 2.2.
3. **Similarity Computation**. The similarity computation between strings is a crucial task for payload-based anomaly detection. With the utilization of vectorial data representations messages can be compared by computing their pairwise distance in the designated geometric space. Similarity measures are explained in Section 2.3.
4. **Anomaly Detection**. In an initial training phase the anomaly detection algorithm learns a global normality model. At detection time a message is compared to the learned data model and based on

its distance an anomaly score is computed. Details on the anomaly detection process can be found in Section 2.4.

2.1 Protocol Analysis

Network protocol analysis is a useful technique to decode and understand data which is encapsulated by an application-specific protocol. Many application-level protocols follow the notion of a common protocol design which is reflected in a unified protocol structure [3]. The majority of application-level protocols stipulate the concept of an application session between two endpoints in which a series of messages is exchanged to accomplish a specific task. Thereby, a *protocol state machine* determines the structure of the application session and specifies legitimate sequences of messages allowed by the protocol. Another essential element of an application protocol is the *message format specification* which defines the structure of an application-layer message. A message format specifies a sequence of fields and their corresponding notion. The syntax of an application-layer protocol can usually be specified by an augmented Backus-Naur-Form which is used to express formal grammars that generate context-free languages. Application protocol analyzer, such as binpac [21], allow to transform re-assembled application-layer messages into a structured data representation, e.g.*parse trees*, which entangle transferred user data with syntactic aspects of the underlying application-layer protocol. In this contribution, we focus on the analysis of message format specifications and do not address the problem of inferring protocol state machines which has been sufficiently addressed in the past. The proposed method is demonstrated using the two application-layer protocols *HTTP* and *RPC* which are explained in detail in the following sections.

2.1.1 Hyper-Text Transfer Protocol

The Hyper-Text Transfer Protocol (HTTP) is one of the most popular text-based application-layer protocols. An example of a typical HTTP request is given below. Control characters are shown as ”.

```
GET /search?q=network+security&gws=ssl&pr=20 HTTP/1.1..
Host: www.google.de..User-Agent: Mozilla/5.0 (X11; ..
Linux x86_64; rv:12.0) Gecko/20100101 Firefox/12.0..
Accept: text/html,application/xhtml+xml,application/
xml;q=0.9,*/*;q=0.8..Accept-Language: en-us,en;q=0.5..
Accept-Encoding: gzip, deflate..Connection: keep-
alive....
```

The GET request contains CGI parameters as well as common HTTP headers. With the protocol specification at hand a protocol analyzer generates a structured representation of the request sequence which is typically realized by parse trees. An example of a corresponding parse tree is shown in Fig. 1(a). The tree consists of non-terminal nodes as well as pre-terminal and terminal nodes. However, due to the limited complexity of the underlying HTTP grammar, relevant information resides at the pre-terminal and terminal level only. Therefore, the tree can be shrunk and converted into a set of key/attribute tuples. Thereby, each pre-terminal node label serves as a unique protocol context key whereas the associated attribute is assembled from connected terminal nodes.

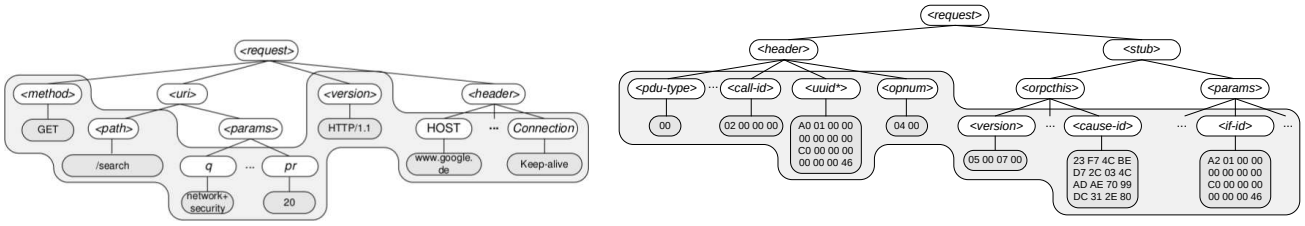
2.1.2 Remote Procedure Calls

A more opaque application-layer protocol is provided by Remote Procedure Call (RPC). A significant part of the Microsoft Windows architecture is composed of services (e.g. DNS, DHCP, DCOM) that communicate with each other in order to accomplish a particular task. Microsoft RPC is a widely used binary application-layer protocol and represents a powerful technology which is utilized by a multitude of services to access functions located at foreign address spaces.
In order to invoke methods remotely, RPC requires to establish a session context. By submitting a `BIND` request the client initiates an RPC session in which the endpoint mapper interface is requested to bind to the desired RPC interface. An example of a `BIND` request is shown below:

```
0000 05 00 0b_A 03 10 00 00 00 78 00 28 00 02 00 00 00
0010 d0 16 d0 16 92 bc 00 00 01 00 00 00 01 00_B 01 00
0020 a0 01 00 00 00 00 00 00 c0 00 00 00 00 00 00 46_C
0030 00 00 00 00 04 5d 88 8a eb 1c c9 11 9f e8 08 00
0040 2b 10 48 60 02 00 00 00_D ...
```

The most important fields of a `BIND` request are highlighted and include protocol data unit type (A) as well as RPC session information. Each session is essentially defined by a context identifier (B) and a universally unique identifier (UUID) which corresponds to the requested RPC interface (C). In order to allow for transfer encoding negotiation, the client provides a coding scheme (D) to the server for each session requested.
The endpoint mapper resolves and returns the endpoint (TCP port) in response to the interface request. Once the client obtains the endpoint it connects to the interface and invokes the desired method by sending a `CALL` request.

```
0000 05 00 00_A 03 10 00 00 00 20 03 00 00 02 00 00 00_E
0010 08 03 00 00 01 00_B 04 00_F 05 00 07 00 01 00 00 00
```



(a) HTTP request parse tree

(b) RPC/DCOM request parse tree (*context identifier is dynamically replaced by the corresponding UUID)

Fig. 1 Generated parse trees representing application-layer protocol requests

```
0020 00 00 00 00 23 f7 4c be d7 2c 03 4c ad ae 70 99
0030 dc 31 2e 80 00 00 00 00 00 00 00 00 00 00 02 00
0040 d8 02 00 00 d8 02 00 00 4d 45 4f 57 04 00 00 00
0050 a2 01 00 00 00 00 00 00 c0 00 00 00 00 00 00 46   G
0060 38 03 00 00 00 00 00 00 c0 00 00 00 00 00 00 46
0070 00 00 00 00 a8 02 00 00 a0 02 00 ...
```

The header essentially specifies a call identifier (E), a session context identifier (B), a method identifier (F) and payload which contains arguments expected by the method. Since the context identifier (B) refers to an active application session in which the client has bound to an interface already the UUID is not explicitly transmitted in a `CALL` request but instead, referenced by the corresponding context identifier.
With the protocol specification at hand the protocol analyzer produces a parse tree which is shown in Fig. 1(b). In this particular example, RPC is used to call the `ISystemActivator` interface in order to request instantiation of a class which is identified by the UUID (G) in the parameter section of the RPC request.
Certainly, method call details can only be extracted from the request if an appropriate RPC stub dissector is in place which is able to analyze RPC payload according to a list of known core interfaces and method declarations. For our considerations, Wireshark's [35] DCE/RPC dissection module is used which allows for concise and automatic parameter value extraction of functions that are declared by well-known RPC interfaces (e.g. LSARPC and SRVSVC).

2.2 Feature Extraction

Application payload is characterized by sequential data which is not applicable for learning methods that operate in metric spaces. Therefore, feature extraction must be performed in order to map sequences into a metric space in which similarity between vectorial representations of sequences can be computed. Formally, a feature map $\phi : \mathcal{X} \mapsto \mathbb{R}^N$ can be defined which maps a data point in the domain of application payloads $\mathcal{X}$ into a N-dimensional metric space - in the following referred to as feature space F - over real numbers:

$$x \longmapsto \phi(x) = (\phi_1(x), \phi_2(x), \ldots, \phi_N(x)), \tag{1}$$

where $\phi_i(x) \in \mathbb{R}_{\geq 0}$ represents the value of the i-th feature. Thereby, the sole choice of the mapping function $\phi(x)$ provides a powerful instrument to transform data into a representation that is suitable for a given problem.
In this section we describe feature mappings based on different types of features. While protocol analysis suggests to extract features from tree structures such as parse trees, the detection of suspicious byte patterns favors the extraction of sequential features. Once a feature space has been designed, there are several feature embeddings to chose from. Common feature embeddings include binary, count as well as frequency representations of individual features.

Syntax Features. The syntactic structure of transferred application-level payload can be extracted by conducting protocol analysis which eventually allows to generate parse trees. An intuitive way to characterize a parse tree structure is to consider each node independently of its syntactic context, i.e. predecessors as well as successors. The following feature map can be used to determine structural similarity between sequences:

$$\phi : s \longmapsto (\phi_\tau(s))_{\tau \in \mathcal{T}^*} \in F \ ,$$

where $\mathcal{T}^*$ denotes the set of all possible unique subtrees. The mapping function $\phi(s)$ is defined as follows:

$$\phi_t(s) = \begin{cases} 1, & t \in \{\tau \in \mathcal{T}^* \mid n(\tau) = 1\} \\ 0, & otherwise, \end{cases} \tag{2}$$

where $n(\tau)$ is a function which returns the number child nodes attached to a node τ. Using this mapping, each dimension in F corresponds to a binary feature indicating the presence of a particular pre-terminal node t in the actual parse tree of a sequence s. For

the rest of this paper, the set of pre-terminal nodes as a representation of an application-level message is referred to as *bag-of-token* features.

Sequential Features. An intuitive data representation at byte-level involves the extraction of unique substrings by moving a sliding window of length n over a sequence. The resulting set of feature strings are called n-grams. Each sequence s is embedded into a n-dimensional metric space F where $F \in \mathbb{R}^n$, using the following feature map:

$$\phi : s \longmapsto (\phi_w(s))_{w \in \Sigma^n} \in F,$$

where Σ^n refers to the set of all possible strings w of length n induced by an alphabet Σ.

Context-aware Sequential Features. Protocol dissection allows to attach syntactic information to sequential features. By introducing a novel data representation, so called contextual n-grams (c_n-grams), syntactic features can be combined with sequential features in an unified feature space using the feature mapping $\phi(s)$ below.

$$\phi : s \longmapsto (\phi_{w,\tau}(s))_{w_\tau \in \Sigma^n} \in F,$$

where Σ^n refers to the set of all possible strings w_τ of length n induced by an alphabet Σ and $\tau \in \mathcal{T}^*$ refers to a subtree in the set of all possible subtrees. A schematic illustration of c_n-grams is shown in Fig. 2.

The c_n-gram data structure allows to efficiently store n-grams along with syntactic labels. Each entry in the data structure has a unique hash value. The hash value encodes both syntactic context and sequential information represents a c_n-gram. The syntactic label information (i.e. pre-terminal nodes from the parse tree) is encoded using the first k-bits of the CPU's register size m, whereas the remaining $m - k$ bits are used to encode the actual n-gram ($n \leq \lfloor \frac{m-k}{8} \rfloor$) observed in the terminal string attached to a pre-terminal node. As a result, a particular n-gram is allowed to be contained in terminal strings attached to different pre-terminal nodes which represents an extension to the regular definition of n-grams outlined in 2.2.

In the example shown in Fig. 2 extracted HTTP pre-terminal nodes are encoded and combined with n-grams from parsed terminal strings represented as c_n-gram. Finally, the set of c_n-grams is convoluted in a joint histogram H.

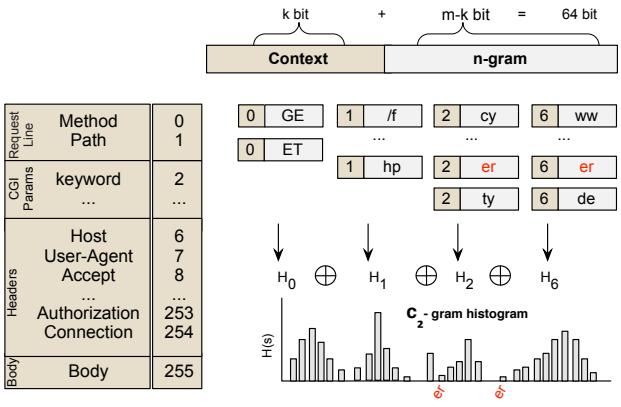


Fig. 2 c_n-grams: context-aware sequential data representation

2.3 Similarity Measure

Once a sequence is mapped into a feature space F a kernel function $k : \mathcal{X}^2 \to \mathbf{R}$ can be applied to determine pairwise similarity between data points $\{x_1, ..., x_n\} \subset \mathcal{X}$. Thereby, the type of kernel function entails an implicit mapping of a data point in F into a possibly even higher dimensional feature space $\mathcal{F}$ which could, in some situations, facilitate the learning process.

In this section we describe two different kernel functions that are most widely used in various application domains, the *Linear Kernel* and the *Radial Basis Function Kernel* (RBF).

Linear Kernel. The linear kernel is defined by a dot product between two vectors x and y and is used to determine similarity between data points which are linearly mapped into $\mathcal{F}$:

$$\begin{aligned} k(x, y) &= \langle \phi(x), \phi(y) \rangle \\ &= \sum_{i=1}^{n} \phi_i(x) \phi_i(y). \end{aligned} \tag{3}$$

With regard to network security, the major benefit of this particular kernel becomes immediately clear. Due to the bijective mapping, a pre-image of every data point in the feature space $\mathcal{F}$ exists which allows to directly deduce differences in features located in F.

Although it seems to quickly become computational unfeasible to compute dot products over sequential features the utilization of efficient data structures such as suffix trees or hash tables allow to compute the similarity $k(x, y)$ in $O(|x| + |y|)$ time [29, 32]. The dot product is of particular mathematical appeal because it provides a geometric interpretation of a similarity score in terms of length of a vector as well as angle and

distance between two vectors. Therefore, the Euclidean distance $d_{eucl}(x, y)$ can be easily derived from the above kernel formulation:

$$\begin{aligned} d_{eucl}(x,y) &= \|x-y\|^2 \\ &= \sqrt{k(x,x)+k(y,y)-2k(x,y)}. \end{aligned} \tag{4}$$

RBF-Kernel. A more complex similarity measure is provided by the RBF-Kernel which implicitly maps data points into a feature space $\mathcal{F}$ which is non-linearly related to the input space. The RBF-kernel is defined as follows:

$$k(x,y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right), \tag{5}$$

where σ controls the width of the gaussian distribution and directly affects the shape of the learner's decision surface. While a large σ results in a linear decision surface which indicates a linearly separable problem, a small value of σ generates a peaky surface which strongly adapts to the distribution of the data in F. The interpretation of RBF-kernel values is non-trivial because, unlike linear kernel functions, an RBF-kernel implicitly maps data points from the input space to an infinite dimensional feature space F. As an example, consider two data points $x, y \in \mathbb{R}^2$, where $x = (x_1, x_2)$, $y = (y_1, y_2)$, the RBF-kernel can be re-formulated as an infinite sum of inner products over features in input space using Taylor series as shown in Eq.(6).

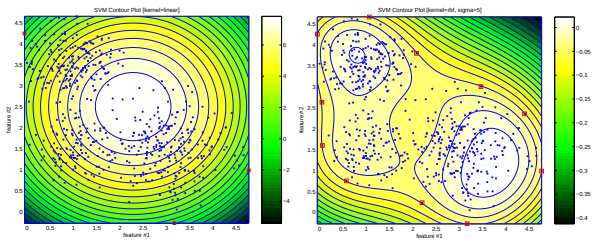
$$\begin{aligned} k(x,y) &= \exp\left(-\|x-y\|^2\right), \\ &= \exp\left(-(x_1-y_1)^2-(x_2-y_2)^2\right), \\ &= \exp\left(-x_1^2+2x_1y_1-y_1^2-x_2^2+2x_2y_2-y_2^2\right), \\ &= \exp\left(-\|x\|^2\right)\cdot\exp\left(-\|y\|^2\right)\cdot\exp\left(2x^Ty\right), \\ &= \exp\left(-\|x\|^2\right)\cdot\exp\left(-\|y\|^2\right)\cdot\textstyle\sum_{n=0}^{\infty}\frac{(2x^Ty)^n}{n!}. \end{aligned} \tag{6}$$

2.4 Anomaly Detection

The problem of anomaly detection can be solved mathematically considering the geometric relationship between vectorial representations of messages. Although anomaly detection methods have been successfully applied to different problems in intrusion detection, e.g. identification of anomalous program behavior [e.g. 7, 8], anomalous packet headers [e.g. 17, 19] or anomalous network payloads [e.g. 12, 25, 26, 33, 34], all methods share the same concept – *anomalies are deviations from a model of normality* – and differ in concrete notions of normality and deviation. For our purpose we use the one-class support vector machine (OC-SVM) proposed in [31] which fits a minimal enclosing hypersphere to the data which is characterized by a center θ and a radius R. Mathematically, this can be formulated as a quadratic programming optimization problem:

$$\begin{aligned} \min_{\substack{R\in\mathbb{R}\\ \xi\in\mathbb{R}^n}} \quad & R^2 + C\sum_{i=1}^{n}\xi_i \\ \text{subject to:} \quad & ||\phi(x_i)-\theta||^2 \le R^2+\xi_i, \\ & \xi_i \ge 0. \end{aligned} \tag{7}$$

By minimizing R^2 the volume of the hypersphere is minimized given the constraint that training objects are still contained in the sphere which can be expressed by the constraint in Eq.(7).



(a) Linear Kernel (b) RBF Kernel

Fig. 3 Anomaly detection using one-class support vector machine with linear and non-linear decision functions. Support vectors are shown with red edging.

A major benefit of this approach is the control of generalization ability of the algorithm [20], which enables one to cope with noise in the training data and thus dispense with laborious sanitization, as proposed by Cretu et al. [2]. By introducing slack variables ξ_i and penalizing the cost function we allow the constraint to be softened. The regularization parameter $C = \frac{1}{N\nu}$ controls the trade-off between radius and errors (number of training points that violate the constraint) where ν can be interpreted as a permissible fraction of outliers in the training data. The solution of the optimization problem shown in Eq. (7) yields two important facts:

1. The center $\theta = \sum_i \alpha_i\phi(x_i)$ of the sphere can be expressed as a linear combination of training points.
2. Each training point x_i is associated with a weight α_i, $0 \le \alpha_i \le C$, which determines the contribution of the i-th data point to the center and reveals information on the location of x_i. If $\alpha_i = 0$ then

x_i lies in the sphere ($||x_i - c||^2 < R^2$) and the data point can be considered as **normal**. In constrast, if $\alpha_i = C$ x_i can be interpreted as an **outlier** ($||x_i - c||^2 > R^2$). In both cases data points are excluded from the model of "normality". Thus, only those training points yielding $0 < \alpha_i < C$ are located on the surface of the sphere ($||\phi(x_i) - \theta||^2 = R^2$) and thus, define the model of normality as illustrated in Fig. 3. These particular points are known as **support vectors**.

3. The radius R which is explicitly given by the solution of the optimization problem in Eq. (7) refers to the distance from the center θ of the sphere to the boundary (defined by the set of support vectors) and can be interpreted as a threshold for a decision function.

Finally, having determined a model of normality the anomaly score S_z for a test data point z can be defined as the distance from the center in the feature space:

$$\begin{aligned} S_z &= ||\phi(z) - \theta||^2 \\ &= \sum_{w \in \mathcal{A}} (\phi_w(z) - \theta_w)^2 \\ &= \sum_{w \in \mathcal{A}} (\phi_w(z) - \sum_{i=1}^{n} \alpha_i \phi_w(x_i))^2 \\ &= k(z,z) - 2\sum_i \alpha_i k(z, x_i) + \sum_{i,j} \alpha_i \alpha_j k(x_i, x_j), \end{aligned} \tag{8}$$

where the similarity measure $k(x, y)$ between two points x and y defines a kernel function as introduced in Section 2.3. Depending on the similarity measure at hand data models of different complexity can be learned. For example, as shown in Fig. 3(a) the application of a linear kernel always results in an uniform hypersphere. Thus, the resulting model provides a rather general description of the data. However, if data happens to follow a multi-modal distribution the risk of absorbing outliers in low density regions of the hypersphere might increase. On the contrary, the utilization of an RBF-kernel allows to adopt the distribution characteristics of the data resulting in more complex data models as shown in Fig. 3(b). Of course, the downside of these kind of measures is their lack of interpretability as data points are implicitly mapped into an infinite dimensional feature space in which the identification of individual feature contributions to the overall dissimilarity between two data points becomes difficult (c.f. Section 2.3).

2.5 Feature Visualization

So far, we have discussed how payloads are extracted and mapped into a geometric space in which anomaly detection is carried out to identify deviations from a previously learned model of normality. At this point the following question might arise: why is a data point considered as an anomaly and what constitutes the anomaly? In this section, we derive a feature visualization for payload-based anomaly detection which allows to trace back an anomaly to individual features in the payload and thus, provide a technique that not only helps to understand the reason for an anomaly but also means to localize suspicious pattern. Geometrically, a feature can be considered relevant if it has a significant impact on the norm of a vector. Consequently, the anomaly score $S(z)$ can be expressed as a composition of individual dimensions of $\mathbb{R}^{|\mathcal{A}|}$ as shown in Eq. 8. We refer to $\delta_z = (\phi_w(x) - \theta)^2_{w \in \mathcal{A}}$ as *feature differences*, an intuitive visualization technique to explore sequential disparity which has been originally introduced to determine discriminating q-grams in network traces [25]. The entries of δ_z reflect the individual contribution of a string feature to the deviation from normality represented by θ.

While feature differences provide sufficient means to visualize anomalous network features, a security practitioner might also be interested to directly inspect the portions of the payload that constitute an anomaly. Therefore, the concept of *feature differences* is incorporated into a method known as *feature shading* [27]. The idea of feature shading is to assign a number $m_j \in \mathbf{R}$ to each position j in a payload reflecting its deviation from normality. As a result, the payload can be overlaid with a color shading according to the amount of deviation at a particular position. Considering the generic definition of string features S, each position j in the payload can be associated with multiple feature strings. Hence, a set M_j can be defined which contains all strings s matching a position j of a payload z:

$$M_j = \{z[i, ..., i + |s|] = s \mid s \in S\} \quad j - k + 1 \leq i \leq j \tag{9}$$

where $z[i, ..., i + |s|]$ denotes a substring of length k in z starting at position i . By using M_j the contribution m_j of position j to an anomaly score can be determined as follows:

$$m_j = \frac{1}{|M_j|} \sum_{s \in M_j} - \theta_s^2. \tag{10}$$

An abnormal pattern located at j corresponds to low frequencies in the respective dimensions of the learned model θ and therefore, results in a small value of m_j. A

frequent string is characterized by high values in θ and yields a high value of m_j.

3 Experimental Results

In this section we present results on experiments involving recorded network traffic. Experiments are designed to investigate the benefits of the proposed data representation, specifically with regard to the problem of detecting unknown attacks. Experiments are carried out off-line on two data sets containing two prominent types of network traffic: HTTP and RPC. In our setup, inbound packets are captured, re-assembled and forwarded to *binpac*, a flexible protocol analyzer which extracts and parses application-layer messages. As a baseline, anomaly detection is carried out on application-layer messages using sequential features, i.e. n-grams, and linear as well as non-linear feature mappings. Our experiments are designed to address the following two questions:

1. To which extent does the choice of data representation affect the detection accuracy of an anomaly detector with regard to different network protocols, attack types and basic detector evasion?
2. How does the c_n-gram data representation help to explain and interpret geometric anomalies?

3.1 Data Sets

We evaluate our method on two sanitized data sets containing traffic recorded on a publicly accessible web server (HTTP traffic) as well in an industrial automation testbed (RPC traffic). For both data sets, application-level messages were extracted using the *binpac* protocol parser. Furthermore, we simulated a broad collection of application-level attacks on the target servers and finally mixed those attacks with the captured application-level messages. Attacks were taken from the Metasploit framework [1] as well as from common security forums such as *securityfocus.com*, *remote-exploit.org* and *xssed.com*. Each attack class consists of multiple attack instances carrying different payloads.

3.1.1 ***BLOG09****: Plain-text Protocol - HTTP*

The first data set comprises a sample of 150000 HTTP requests recorded on a blog site web server accessible from the internet over a period of ten days. With an average request length of 389 bytes the size of an HTTP request ranges between 39 and 61127 bytes which is mainly due to the presence of a notable fraction of web spam in the corpus. Protocol analysis reveals 317 distinct tokens including common HTTP protocol elements as well as CGI parameters. In average, each request consists of nine tokens whereas the average length of a corresponding attribute is about 57 bytes. Attacks were chosen to cover a variety of attack types including *buffer overflows* (BUF), *cross site scriptings* (XSS), *SQL injections* (SQL) and *command injections* (CI). Details on the attacks can be found in Table 1. In order to put forward a realistic setup, the majority of attacks was customized to fit existing vulnerabilities. For the rest of the paper this data set is referred to as *BLOG09-I*. Moreover, in order to expose differences in the detection behavior of an anomaly detector based on various data representations we created a second, even larger attack set (*BLOG09-II*) in which the attacks from the original attack set were tuned to increase structural and sequential similarity to normal traffic to impede attack detection. Therefore, HTTP header keys and values frequently observed in normal traffic were added to the attack instances contained in *BLOG09-II*:

```
Host: tim.xxxxxxxx.de..User-Agent: Mozilla/5.0(X11; U
;Linux x86_64; de; rv:1.9.0.7) Gecko/2009030423 Ubunt
u/8.10 (intrepid) Firefox/3.0.7..Accept: text/html,ap
plication/xhtml+xml,application/xml;q=0.9,*/*;q=0.8..
Accept-Language: de-de,de;q=0.8,en-us;q=0.5,en;q=0.3.
.Accept-Encoding: gzip,deflate..Accept-Charset: ISO-8
859-1,utf-8;q=0.7,*;q=0.7..Connection: close..
```

Class	CVE	n	Description	Type
1	-	5	wp_profile applet	XSS
2	-	5	wp_profile embed	XSS
3	-	5	wp_profile iframe	XSS
4	-	5	wp_profile body	XSS
5	-	11	wp_profile* crlf	XSS
6	2005-1810	8	wp_index 1=1	SQL
7	2005-1810	5	wp_index outfile	SQL
8	2008-1982	5	wp_index benchmark	SQL
9	2008-4008	11	weblogic trans_enc	BUF
10	2009-0921	9	openview OvOSLocale	BUF
11	2004-1134	8	iis (isapi/w3who)	BUF
12	2001-0241	10	iis (isapi/printing)	BUF
13	2004-0798	10	ipswitch maincfgret	BUF
14	2004-0798	2	edirectory_host	BUF
15	2005-2848	3	barracuda_img_exec	CI

Table 1 HTTP attacks

3.1.2 ***AUT09****: Binary Protocol - RPC*

The second data set consists of 765000 TCP packets that were captured in an industrial automation testbed

[1] http://www.metasploit.com

over a period of eight hours. The traffic contains mostly DCE/RPC traffic. Details on the attack set can be found in Table 2.

Class	CVE	n	Description	Type
1-3	2003-0352	3	RPC/DCOM	BUF
4	2005-0059	1	RPC MSMQ	BUF
5	2007-1748	1	RPC DNS	BUF
6	2003-0533	1	LSASS	BUF
7-11	2006-3439	5	SRVSVC	BUF
12-14	2008-4250	3	SRVSVC	BUF

Table 2 DCE/RPC attacks

3.2 Validation Metrics

Since feature extraction and anomaly detection may require certain parameters to be set our experiments are casted into a cross validation framework in order to determine a parameter configuration which maximizes the detection accuracy for the feature representation in question. To this end, data is split into three distinct partitions for *training*, *validation*, and *testing*. Since we focus on the detection of unknown attacks validation and test partitions contain attacks taken from distinct attack classes. Each model is learned using a sample of 1000 normal messages drawn from the training partition and subsequently validated on 10 distinct validation samples. A validation sample also consists of 1000 normal messages mixed up with attacks from the validation partition. Finally, the model that maximizes the detection accuracy during the validation phase is applied on a test sample. The detection accuracy is measured in terms of area under *receiver operating characteristic* curve ($ROC_{0.01}$) which integrates true positive values over the false positive interval $[0, 0.01]$. For statistical reasons experiments are repeated and results are averaged over 20 repetitions.

3.3 Impact of Data Representation on Detection Accuracy

In this experiment we investigate the impact of different data representations on the detection accuracy with regard to unknown attacks.

BLOG09-I . At first, we present experimental results on the *BLOG09-I* data set. Table 3 shows the model parameters that yield the highest detection accuracy during validation grouped by similarity measure and feature type.

Although the c_n-gram data representation clearly outperforms the bag-of-token features the highest detection rate is achieved by conventional n-grams. The choice of a small n-gram length indicates a comparably modest anomaly detection problem. This is not surprising, since the majority of attacks in the *BLOG09-I* data set is predominantly represented by an attack vector and the respective payload which suggests strong structural as well as sequential dissimilarity to normal traffic. Furthermore, experiments with the RBF-kernel demonstrate a detection behavior which is comparable to the Linear Kernel.

Measure	Feature	n	Embedding	AUC
linear	n-grams	2	bin	0.99 $\pm$0.01
rbf_{10-100}	n-grams	2	bin	0.99 $\pm$0.01
linear	c_n-grams	2	freq	0.92 $\pm$0.01
linear	bag-of-token	-	bin	0.85 $\pm$0.06

Table 3 Area under $\text{ROC}_{0.01}$-curve (AUC) of best models during validation over HTTP requests (*BLOG09-I*) grouped by measure and feature

Given the parameter settings in Table 3 data models are learned and finally applied on separate test samples. The results are depicted in Fig. 4(a) which shows a comparison of ROC-curves that result from the application of different data representations. Similar to the validation results c_n-grams clearly outperform bag-of-token features while n-grams prevail over all data representations. A more detailed analysis of the results is presented in Table 6 which outlines individual false positive rates per attack class (percentage of "normal" test points having an anomaly score above the smallest score achieved by instances of a particular attack class). While the bag-of-token approach suffers false positives around 1% for a few attack classes the detection performance of n-grams and c_n-grams can be considered comparable. However, a major difference in the detection behavior arises for attack class 6 which comprises short and relatively harmless SQL injections of the form "' or 1=1 - -" and which are primarily applied to escalate SQL selection statements.

Results clearly show that the utilization of c_n-grams in that particular case does not provide significant advantages over conventional n-gram data representations. The reason for the poor detection accuracy of that particular attack class resides in the fact, that the amount of suspicious characters make up only a very small contribution to the overall geometric norm of the attack data point embedded

in the c_n-gram feature space which is significantly larger than the conventional n-gram feature space. Therefore, the detection accuracy using the n-gram data representation is significantly higher.

BLOG09-II . We now present results on the *BLOG09-II* data set in which attacks are tuned to match normal traffic characteristics. A list of best models selected during validation is shown in Table 4. In contrast to the validation results on *BLOG09-I* the best model based on n-grams requires a larger string length which suggests a more difficult anomaly detection problem caused by the sole adherence of HTTP protocol headers to the attack vector. As a consequence, the detection accuracy deteriorates for all data representations in comparison to the results of the *BLOG09-I* validation results. As expected, the bag-of-token features completely fail to provide sufficient discriminating information on the attacks yielding a detection accuracy of less than 10% within a false positive interval of [0,0.01]. Interestingly, the data representation providing the highest overall detection accuracy changes for both experiments. While in the *BLOG09-I* experiments the n-gram data representation slightly prevails c_n-grams clearly outperform all other representations in the *BLOG09-II* experiments. Moreover, the utilization of protocol information in combination with sequential features favors the choice of a much smaller string length n while strongly improving the expressiveness of the c_n-gram features.

Measure	Feature	n	Embedding	AUC
linear	c_n-grams	2	freq	0.74 ±0.08
linear	n-grams	4	bin	0.44 ±0.07
rbf_{10-100}	n-grams	4	bin	0.44 ±0.07
linear	bag-of-token	-	bin	0.10 ±0.05

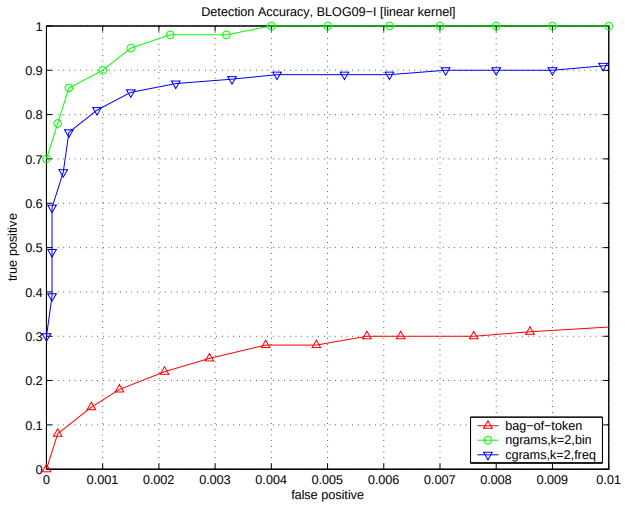
Table 4 Area under $ROC_{0.01}$-curve (*AUC*) of best models during validation over HTTP requests (*BLOG09-II*) grouped by measure and feature

A comparison of ROC-curves for the *BLOG09-II* test data set is depicted in Fig. 4(b). The corresponding false positive rates per attack class are provided in Table 7. While most of the buffer overflows (classes 9-14) can be reliably detected by both n-grams and c_n-grams the utilization of the c_n-gram data representation results in a superior detection accuracy for cross site scripting attacks (classes 1-5) and SQL injections (classes 7-8) which, on the other hand, induces a significant increase in false positives for n-grams. As shown in Fig. 4(b) the incorporation of protocol information into sequential features results in a detection rate of 80% at a false positive level of 1% which is almost twice as high as attained by conventional n-grams (43%).The false positive analysis reveals that by using c_n-grams the accuracy can be strongly improved for majority of non-overflow attacks which is reflected by a major reduction of the respective false positive rate to a level less than 1%.

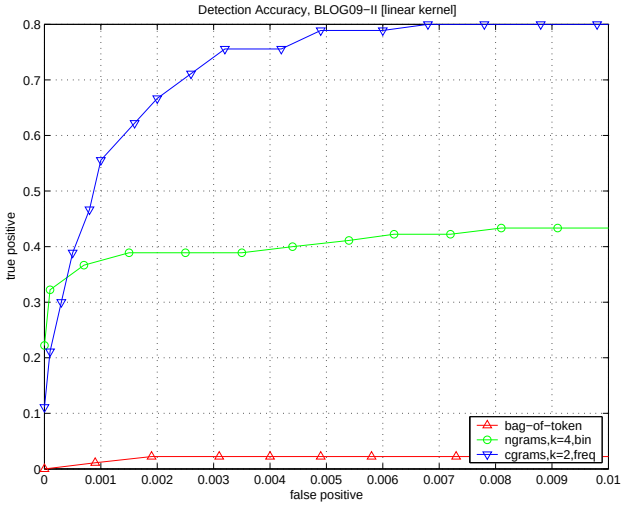
The reason for c_n-grams to outperform conventional n-grams becomes obvious by picturing document frequency differences between normal data and individual attack instances which is shown in Fig 5. The plot displays 1-gram frequency differences between 10000 normal HTTP requests and two different attacks, a buffer overflow (class 14) and a SQL union injection (class 8). A frequency difference of 1 arises from bytes that solely appear in normal data points whereas bytes that are exclusively found in the attack yield a frequency difference of -1. Bytes with a frequency difference close to zero do not provide discriminating information. In the upper two plots frequency differences of both attacks are shown that result from conventional analysis at request-level, whereas the lower two plots show frequency differences of both attacks from token-level analysis. The boxes in the plots mark areas with the largest amount of differences between frequency difference distributions attained from request-level and token-level analysis.

The buffer overflow can be easily detected due to the presence of a large amount of non-printable characters (a large number of points at -1 for byte values in the range of [0,45] and [128,255]). On the other hand, the SQL injection contains mostly ASCII characters, which is reflected by close to zero frequency differences for most of the printable characters. As a consequence, the detection of this kind of attack is comparably difficult using conventional n-gram features. On the contrary, the lower two charts show frequency differences for the same attacks but with regard to the respective vulnerability. The frequency differences of the buffer overflow become even more obvious considering the local byte distribution in the scope of the exploited parameter "Host". Similar clarification takes place for the SQL injection for which many previously normal bytes become clearly anomalous taking the respective vulnerable protocol element ("cat") into account. This results in a major improvement of the detection accuracy.

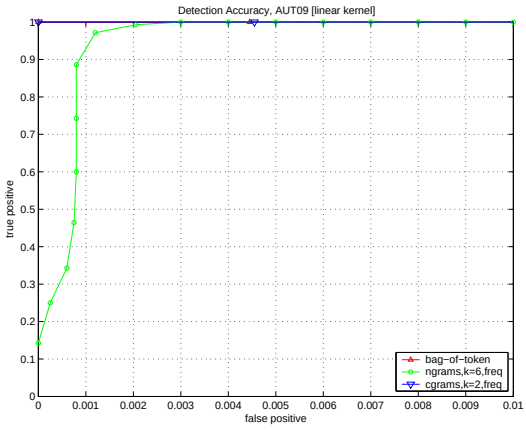
In contrast to the results on the *BLOG09-I* data set which demonstrate insignificant advantages of c_n-grams over conventional n-grams (with regard to the problem of anomaly detection) experiments on the *BLOG09-II* data set show that the utilization of c_n-



(a) HTTP attack set I



(b) HTTP attack set II



(c) RPC attack set

Fig. 4 $ROC_{0.01}$-curve for attacks over binary and text-based application-layer protocols

grams results in a major improvement of detection accuracy, predominantly for web application attacks such as SQL injections and XSS attacks. This is mainly due to the fact that both attack types employ portions of the alphabet that are common with normal traffic. However, since buffer overflows exhibit some kind of binary shell code, the detection accuracy remains largely unaffected for those type of attacks. The results also suggest that in presence of payload customization (e.g. adding frequently occurring HTTP headers) the c_n-gram data representation is more effective than n-grams because the protocol context of a particular n-gram becomes increasingly important if normal and outlier classes are generated by similar alphabets and distributions.

AUT09 . We now present results of experiments conducted on binary network traffic. Table 5 shows the model parameters that yield the highest detection accuracy during validation grouped by similarity measure and feature type. Since multiple best parameter configurations are selected final experiments are carried out using parameters that yield the least complex data model. The results on the test data set are depicted in Fig. 4(c). In contrast to the experiments on HTTP, RPC attacks are perfectly detected using both bag-of-token features and c_n-grams. However, as shown in Table 8 the utilization of c_n-grams demonstrates only a marginal improvement in the detection accuracy compared to n-grams.

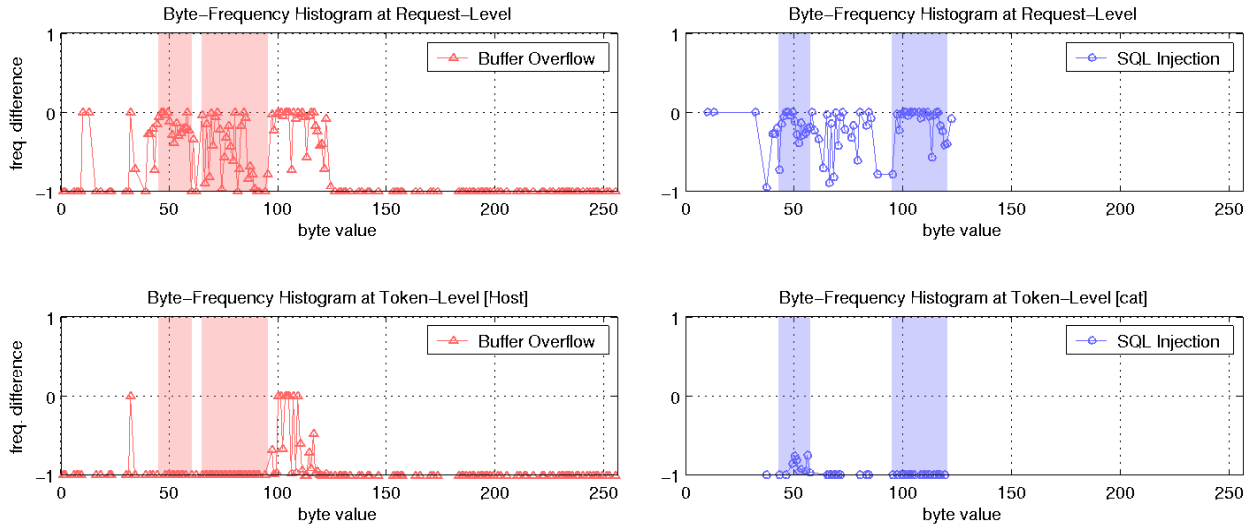


Fig. 5 Byte-frequency differences at request and token level over *BLOG09-II* requests

Measure	Feature	n	Embed	AUC
linear	bag-of-token	-	bin	1
linear	c$_n$-grams	2-4	bin/freq	1
linear	n-grams	4-6	freq	0.99 ±0.01
rbf$_{10-100}$	n-grams	4-6	freq	0.99 ±0.01

Table 5 Area under ROC$_{0.01}$-curve (*AUC*) of best models during validation over RPC requests (AUT09) grouped by measure and feature

3.4 Localization of Anomalous Payloads Using Syntax-Sensitive Features

The c$_n$-gram data representation is not only effective for the detection of web application attacks in network traffic, it can also be used to pinpoint vulnerabilities in protocols and web applications. In this section we demonstrate the use of feature shading, an intuitive method to explain geometric anomalies based on feature differences (c.f. Section 2.5) in presence of protocol context information.

The c$_n$-grams data representation not only allows to track down suspicious byte patterns, it also provides a security operator with information on the syntactic context, e.g. a potentially vulnerable protocol element such as an HTTP header or a CGI parameter. This constitutes a major benefit over conventional n-grams. Geometrically, a protocol element can be considered potentially vulnerable if its associated attribute consists of n-grams which contribute to a large extent to the norm of the overall feature differences vector. An example for a SQL injection is shown in Fig. 6 which displays the contribution to the overall norm as a function of individual protocol elements. The CGI parameter "cat" clearly exhibits the largest fraction of the overall feature differences norm and therefore, might provide substantial information on the attack vector.

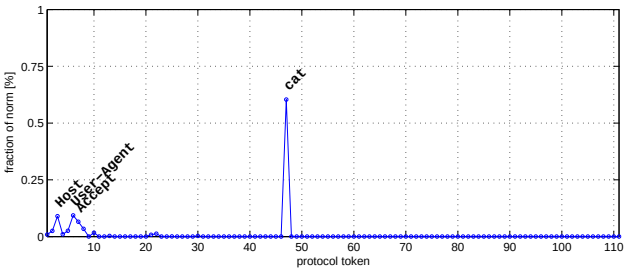


Fig. 6 Contribution of protocol tokens to overall norm of the feature differences vector between 10000 normal HTTP requests and a SQL injection

An example of a corresponding feature shading is given in Fig. 7. For each byte position the corresponding term frequency difference is calculated and graphically highlighted using some coloring scheme. Darker regions exhibit strong frequency differences and thereby, indicate unusual byte patterns. As expected, the most suspicious and longest contiguous byte pattern is located in the attribute of the CGI parameter "cat". In this particular example the attacker mounts a SQL union injection with the objective to create a file on the web server that takes a URI as an argument to eventually allow for remote file inclusion.

The identification of anomalous payloads transferred over HTTP is not very difficult due to the nature of text-based protocols. However, the localization

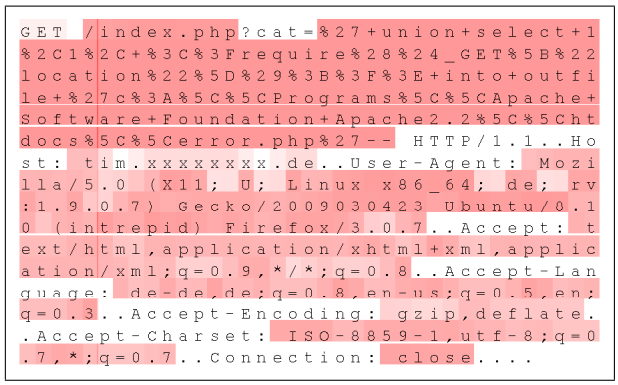


Fig. 7 Feature shading of a SQL union injection based on c_n-grams

of malicious byte patterns transferred over binary protocols is more challenging because the structure of the underlying protocol is not always obvious and typical shell code bytes can also be found in normal traffic. Below is an example for the identification of a potential vulnerability exploited by a RPC buffer overflow.

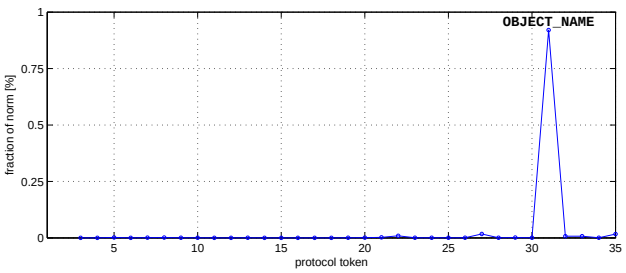


Fig. 8 Contribution of protocol tokens to overall feature differences norm between 10000 normal RPC requests and a RPC buffer overflow

With 93% the largest fraction of the overall feature differences norm is caused by the attribute associated with the token `Object_Name`. The feature shading is presented in Fig. 9 and clearly reveals a long, contiguous byte sequence which according to the opaque coloring suggests anomalous content.

Examination of the corresponding parse tree shows that the attacker binds to the Microsoft Windows remote activation interface registered by the UUID **4d9f4ab8-7d1c-11cf-861e-0020af6e7c57** and exploits a vulnerability located in the *RemoteActivation* function. Thereby, the protocol token `Object_Name` which clearly stands out in Fig. 8 corresponds to a parameter of the *RemoteActivation* function which is overflown by the attacker. Needless to say that the attack vector fits

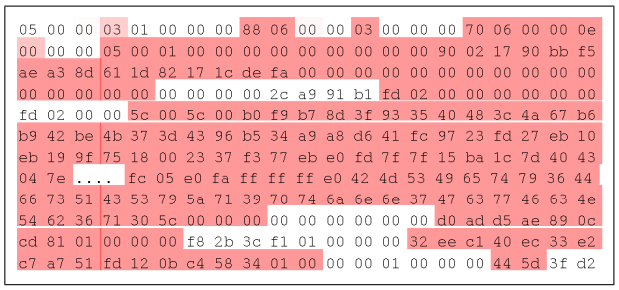


Fig. 9 Feature shading of a RPC buffer overflow attack based on c_n-grams

the well-known RPC/DCOM vulnerability (CVE 2003-0352).

As demonstrated in this section, protocol analysis along with payload-based anomaly detection allows not only to identify suspicious byte patterns in application-layer messages, it also provides a better understanding of attacks and helps to pinpoint potential vulnerabilities.

4 Related Work

Payload-based Anomaly Detection. Over the last decade, a multitude of anomaly-based intrusion detection systems have been proposed which analyze the payload of a packet with regard to sequential anomalies. Wang et al. proposed *Payl* [33] which constructs a simple packet-length specific model of normal traffic. The model relies on the computation of n-gram frequencies ($n \leq 2$) in a payload. The anomaly score is defined by the simplfied Mahalanobis distance to the previously learned n-gram distribution. In a continuitive work, the authors presented *Anagram* [34] which computes a model of normal as well as malicious traffic based on distinct binarized n-grams stored in different Bloom filters. Packets are scored by counting the number of unobserved normal n-grams as well as observed malicious n-grams. An extensive experimental evaluation of geometric outlier detection is provided by Rieck and Laskov [26] who investigated global and local outlier detection methods with regard to the problem of unknown attack detection. Perdisci et al. proposed *McPad* [24] which relies on an ensemble of One-Class SVM detectors. Their approach allows to incorporate different types of features by combining individual detectors in order to achieve a better trade-off between detection accuracy and false positive rate.

Feature Extraction. A large amount of previous work in the domain of network intrusion detection systems has focused on features derived from network and transport layer protocols. An example of such features can be

found in the data mining approach of Lee and Stolfo [15], containing packet, connection and time window features derived from IP and TCP headers. The same work has pioneered the use of "content" features that comprised selected application-level properties such as number of shell prompts, number of failed login prompts, etc. deemed to be relevant for detection of specific attacks. Similar features comprising selected keywords from application-layer protocols have been used by Mahoney and Chan for anomaly detection [18].

General content-based features using the payload distribution of specific byte groups have been first proposed by Kruegel et al. [13] in the context of service-specific anomaly detection using separate normality models for different application-layer protocols. Full distributions of byte values have been considered by Wang and Stolfo [33], eventually extended to models of various languages that can be defined over byte sequences, e.g. n-grams [25, 34]. However, the problem of modeling n-grams is ill-posed because it entails the estimation of distributions in exponentially growing sample spaces. To overcome this problem, Perdisci et al. [24] suggested to use gappy n-grams [16] which effectively reduces the sample space for higher-order n-grams ($n > 2$). Instead of explicitly modeling n-grams, the *Spectrogram* detector [30] which was specifically designed to protect web applications against malicious user input learns a probabilistic representation of legitimate web-layer input using a mixture of factorized n-gram Markov models. Their approach casts n-gram observations into products of conditional probabilities representing transitions between individual characters which eventually results in a linearization of the space complexity.

The incorporation of protocol features into the detection process has been first realized in signature-based IDS. Robust and efficient protocol parsers have been developed for the Bro IDS [22]; however, until recently they were tightly coupled with Bro's signature engine, which has prevented their use in other systems. The development of a stand-alone and extendable protocol parser binpac [21] has provided a possibility for combining protocol analysis with other detection techniques. Incremental and bi-directional parsing as well as error recovery are especially attractive features of binpac. Similar properties at a more abstract level are exhibited by the protocol analyzer GAPAL [1]. Although, protocol parsing by definition limits the scope of analysis to known network protocols unknown protocols can be explored using sophisticated protocol reverse engineering techniques [3, 36].

However, the incorporation of protocol analysis into anomaly detection remains largely unexplored. Kruegel and Vigna [12] have developed a highly effective system for the detection of web attacks by considering separate models for HTTP requests. The system combines models built over specific features such as length, character distribution and request token order defined for individual web applications associated with particular URI paths. Duessel et al. [5] proposed a method which allows to integrate protocol analysis in payload-based anomaly detection based on composite kernel measures. Their experiments showed that significant improvements in the detection accuracy can be achieved, especially for unknown web application attacks. Our method advances this approach in that it allows to transparently map byte sequences into a unique geometric space which reflects sequential as well as syntactical features. Hence, instead of calculating multiple kernels the c$_n$-gram data structure eventually requires to compute one kernel only.

Evasion. Attack obfuscation is a common practice among attackers to avoid detection. *Blending-based* evasion of byte frequency-based network anomaly IDS has been first addressed by Kolesnikov and Dagon [11] who suggest to blend malicious attack packets with normal traffic using advanced obfuscation techniques such as spectrum analysis [4]. Although Fogla et al. [6] present a proof of NP-hardness of the problem a comprehensive experimental evaluation reveals practical feasibility of blending-based attacks in a continuative study. Resistance of payload-based online anomaly detectors against *poisoning-based* evasion schemes has been investigated by Kloft and Laskov [10].

5 Conclusion

In this contribution we propose a general method that facilitates the combination of protocol analysis and payload-based anomaly detection. To this end, we present a novel data representation, so called c$_n$-grams, that allows to integrate protocol features and sequential features in an unified geometric feature space.

We conduct extensive experiments on recorded network traffic featuring both text-based and binary application-layer protocols which demonstrate superior accuracy on the detection of unknown attacks. Our proposition shows that novel attacks can be identified reliably and the detection accuracy can be boosted from 44% using unsupervised anomaly detection with plain sequential features to 80% using combined features in the presence of attack obfuscation. Although the additional effort of protocol analysis does not seem to pay off for simple SQL injections the method

proves to be especially useful for the detection of web application attacks such as XSS and SQL injections in the presence of attack obfuscation by payload customization. Moreover, we show how c_n-grams can be used to explain geometric anomalies to security experts and also provide insight into vulnerabilities by identifying and pinpointing meaningful features in a payload stream using the feature shading technique.

Due to its general nature, the proposed feature extraction method can be applied on any protocol for which a protocol analyzer is available. While in this contribution we focus on the utilization of protocol features derived from message format specifications, future work should address the problem of how to incorporate protocol state machines into sequential feature representations and investigate the identified limitation of both data representations regarding the accurate detection of local anomalies in sparse feature spaces.

References

1. N. Borisov, D.J. Brumley, H. Wang, J. Dunagan, P. Joshi, and C. Guo. Generic application-level protocol analyzer and its language. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2007.
2. G. Cretu, A. Stavrou, M. Locasto, S.J. Stolfo, and A.D. Keromytis. Casting out demons: Sanitizing training data for anomaly sensors. In *ieeesp*, 2008.
3. W. Cui, J. Kannan, and H.J. Wang. Discoverer: automatic protocol reverse engineering from network traces. In *SS'07: Proceedings of 16th USENIX Security Symposium on USENIX Security Symposium*, pages 1–14, 2007.
4. T. Detristan, T. Ulenspiegel, Y. Malcom, and M. Underduk. Polymorphic shellcode engine using spectrum analysis. Phrack Issue 0x3d, 2003.
5. P. Düssel, C. Gehl, P. Laskov, and K. Rieck. Incorporation of application layer protocol syntax into anomaly detection. In *ICISS*, pages 188–202, 2008.
6. P. Folga, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee. Polymorphic blending attacks. In *Proc. of USENIX Security Symposium*, pages 241–256, 2006.
7. S. Forrest, S.A. Hofmeyr, A. Somayaji, and T.A. Longstaff. A sense of self for unix processes. In *Proc. of IEEE Symposium on Security and Privacy*, pages 120–128, Oakland, CA, USA, 1996.
8. D. Gao, M.K. Reiter, and D. Song. Behavioral distance measurement using hidden markov models. In *Recent Adances in Intrusion Detection (RAID)*, pages 19–40, 2006.
9. K.L. Ingham, A. Somayaji, J. Burge, and S. Forrest. Learning DFA representations of HTTP for protecting web applications. *Computer Networks*, 51(5):1239–1255, 2007.
10. M. Kloft and P. Laskov. Security analysis of online centroid anomaly detection. Technical Report UCB/EECS-2010-22, EECS Department, University of California, Berkeley, Feb 2010.
11. O. Kolesnikov, D. Dagon, and W. Lee. Advanced polymorphic worms: Evading IDS by blending with normal traffic. In *Proc. of USENIX Security Symposium*, 2004.
12. C. Kruegel and G. Vigna. Anomaly detection of web-based attacks. In *Proc. of 10th ACM Conf. on Computer and Communications Security*, pages 251–261, 2003.
13. C. Kruegel, T. Toth, and E. Kirda. Service specific anomaly detection for network intrusion detection. In *Proc. of ACM Symposium on Applied Computing*, pages 201–208, 2002.
14. C. Kruegel, G. Vigna, and W. Robertson. A multi-model approach to the detection of web-based attacks. *Computer Networks*, 48(5), 2005.
15. W. Lee and S.J. Stolfo. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information Systems Security*, 3:227–261, 2000.
16. H. Lodhi, C. Saunders, J. Shawe-Taylor, N. Cristianini, and C. Watkins. Text classification using string kernels. *Journal of Machine Learning Research*, 2:419–444, 2002.
17. M.V. Mahoney and P.K. Chan. PHAD: Packet header anomaly detection for identifying hostile network traffic. Technical Report CS-2001-2, Florida Institute of Technology, 2001.
18. M.V. Mahoney and P.K. Chan. Learning nonstationary models of normal network traffic for detecting novel attacks. In *Proc. of ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*, pages 376–385, 2002.
19. M.V. Mahoney and P.K. Chan. An analysis of the 1999 DARPA/Lincoln Laboratory evaluation data for network anomaly detection. In *Recent Adances in Intrusion Detection (RAID)*, pages 220–237, 2004.
20. K.-R. Müller, S. Mika, G. Rätsch, K. Tsuda, and B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Neural Networks*, 12(2):181–201, May 2001.
21. R. Pang, V. Paxson, R. Sommer, and L.L. Peterson. binpac: a yacc for writing application protocol parsers. In *Proc. of ACM Internet Measurement Conference*, pages 289–300, 2006.
22. V. Paxson. Bro: a system for detecting network intruders in real-time. In *Proc. of USENIX Security Symposium*, pages 31–51, 1998.
23. V. Paxson. The bro 0.8 user manual. Lawrence Berkeley National Laboratroy and ICSI Center for Internet Research, 2004.
24. R. Perdisci, D. Ariu, P. Fogla, G. Giacinto, and W. Lee. McPAD: A multiple classifier system for accurate payload-based anomaly detection. *Computer Networks*, In Press, Corrected Proof:–, 2008.
25. K. Rieck and P. Laskov. Detecting unknown network attacks using language models. In *Detection of Intrusions and Malware, and Vulnerability Assessment, Proc. of 3rd DIMVA Conference*, LNCS, pages 74–90, July 2006.
26. K. Rieck and P. Laskov. Language models for detection of unknown attacks in network traffic. *Journal in Computer Virology*, 2(4):243–256, 2007.
27. K. Rieck and P. Laskov. Visualization and explanation of payload-based anomaly detection. In *Proc. of European Conference on Computer Network Defense (EC2ND)*, 2009.
28. M. Roesch. Snort: Lightweight intrusion detection for networks. In *Proc. of USENIX Large Installation System Administration Conference LISA*, pages 229–238, 1999.
29. J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge University Press, 2004.
30. Y. Song, A.D. Keromytis, and S.J. Stolfo. Spectrogram: A mixture-of-markov-chains model for anomaly detection in web traffic. In *Proc. of Network and Distributed System Security Symposium (NDSS)*, 2009.

31. D. Tax and R. Duin. Data domain description by support vectors. In M. Verleysen, editor, *Proc. ESANN*, pages 251–256, Brussels, 1999. D. Facto Press.
32. S.V.N. Vishwanathan and A.J. Smola. Fast kernels for string and tree matching. In K. Tsuda, B. Schölkopf, and J.F. Vert, editors, *Kernels and Bioinformatics*, pages 113–130. MIT Press, 2004.
33. K. Wang and S.J. Stolfo. Anomalous payload-based network intrusion detection. In *Recent Adances in Intrusion Detection (RAID)*, pages 203–222, 2004.
34. K. Wang, J.J. Parekh, and S.J. Stolfo. Anagram: A content anomaly detector resistant to mimicry attack. In *Recent Adances in Intrusion Detection (RAID)*, pages 226–248, 2006.
35. Wireshark. Wireshark: Network protocol analyzer. Internet: http://www.wireshark.org, Accessed: 2010.
36. G. Wondracek, P. Milani Comparetti, C. Kruegel, and E. Kirda. Automatic Network Protocol Analysis. In *15th Symposium on Network and Distributed System Security (NDSS)*, 2008.

A Experiment Results Details

Type	Attack Class	**linear**, bag-of-token		**linear**, n-**grams**		**rbf**, n-**grams**		**linear**, cn-**grams**	
		FP_{avg}	FP_{std}	FP_{avg}	FP_{std}	FP_{avg}	FP_{std}	FP_{avg}	FP_{std}
XSS	1	0.0187	±0.0077	0.0000	±0.0000	0.0000	±0.0000	0.0000	±0.0000
	2	0.0137	±0.0074	0.0000	±0.0000	0.0000	±0.0000	0.0000	±0.0000
	3	0.0140	±0.0042	0.0000	±0.0000	0.0000	±0.0000	0.0001	±0.0004
	4	0.0150	±0.0070	0.0007	±0.0008	0.0007	±0.0008	0.0005	±0.0012
	5	0.0000	±0.0000	0.0000	±0.0000	0.0000	±0.0000	0.0000	±0.0000
SQL	6	0.0152	±0.0053	0.0027	±0.0015	0.0027	±0.0015	0.6820	±0.0478
	7	0.0118	±0.0015	0.0000	±0.0000	0.0000	±0.0000	0.0000	±0.0000
	8	0.0147	±0.0057	0.0000	±0.0000	0.0000	±0.0000	0.0000	±0.0000
BUF	9	0.0150	±0.0072	0.0000	±0.0000	0.0000	±0.0000	0.0000	±0.0000
	10	0.0010	±0.0007	0.0000	±0.0000	0.0000	±0.0000	0.0000	±0.0000
	11	0.0139	±0.0068	0.0000	±0.0000	0.0000	±0.0000	0.0000	±0.0000
	12	0.0003	±0.0005	0.0000	±0.0000	0.0000	±0.0000	0.0001	±0.0004
	13	0.0000	±0.0000	0.0000	±0.0000	0.0000	±0.0000	0.0000	±0.0000
	14	0.1698	±0.0438	0.0000	±0.0000	0.0000	±0.0000	0.0000	±0.0000
CI	15	0.0037	±0.0022	0.0000	±0.0000	0.0000	±0.0000	0.0000	±0.0000

Table 6 False positive rates for unknown HTTP attacks (*BLOG09-I*) on test data

Type	Attack Class	**linear**, bag-of-token		**linear**, n-**grams**		**rbf**, n-**grams**		**linear**, cn-**grams**	
		FP_{avg}	FP_{std}	FP_{avg}	FP_{std}	FP_{avg}	FP_{std}	FP_{avg}	FP_{std}
XSS	1	0.4044	±0.0858	0.0802	±0.0301	0.0798	±0.0316	0.0004	±0.0005
	2	0.4250	±0.0000	0.1260	±0.0000	0.1260	±0.0000	0.0000	±0.0000
	3	0.4248	±0.0686	0.2683	±0.0523	0.2682	±0.0511	0.0017	±0.0014
	4	0.3750	±0.0516	0.1223	±0.0154	0.1220	±0.0166	0.0007	±0.0006
	5	0.4452	±0.0637	0.2074	±0.0502	0.2078	±0.0505	0.0012	±0.0008
SQL	6	0.4357	±0.0796	0.4143	±0.0291	0.4143	±0.0289	0.9263	±0.0159
	7	0.6779	±0.1752	0.1333	±0.0356	0.1330	±0.0358	0.0034	±0.0020
	8	0.4697	±0.1048	0.0389	±0.0160	0.0386	±0.0158	0.0019	±0.0012
BUF	9	0.4113	±0.0660	0.0000	±0.0000	0.0000	±0.0000	0.0000	±0.0000
	10	0.1677	±0.0347	0.0000	±0.0000	0.0000	±0.0000	0.0000	±0.0000
	11	0.4054	±0.0489	0.0000	±0.0000	0.0000	±0.0000	0.0001	±0.0004
	12	0.1274	±0.0728	0.0000	±0.0000	0.0002	±0.0004	0.0012	±0.0008
	13	0.0005	±0.0007	0.0000	±0.0000	0.0000	±0.0000	0.0005	±0.0007
	14	0.8547	±0.0412	0.0000	±0.0000	0.0000	±0.0000	0.0001	±0.0004
CI	15	0.5048	±0.0758	0.3335	±0.0784	0.3318	±0.0757	0.0008	±0.0010

Table 7 Average false positive rates for unknown HTTP attacks (*BLOG09-II*) on test data

Attack Class	**linear**, bag-of-token		**linear**, n-**grams**		**rbf**, n-**grams**		**linear**, c$_n$-**grams**	
	FP_{avg}	FP_{std}	FP_{avg}	FP_{std}	FP_{avg}	FP_{std}	FP_{avg}	FP_{std}
1	0	0	0.0010	±0.0012	0.0010	±0.0012	0	0
2	0	0	0.0012	±0.0010	0.0012	±0.0010	0	0
3	0	0	0.0008	±0.0008	0.0008	±0.0008	0	0
4	0	0	0	0	0	0	0	0
5	0	0	0	0	0	0	0	0
6	0	0	0	0	0	0	0	0
7	0	0	0.0008	±0.0009	0.0001	±0.0003	0	0
8	0	0	0.0008	±0.0010	0.0008	±0.0009	0	0
9	0	0	0.0008	±0.0007	0.0009	±0.0011	0	0
10	0	0	0.0006	±0.0005	0.0006	±0.0005	0	0
11	0	0	0.0008	±0.0008	0.0008	±0.0008	0	0
12	0	0	0.0006	±0.0007	0.0006	±0.0007	0	0
13	0	0	0.0009	±0.0008	0.0009	±0.0008	0	0
14	0	0	0.0004	±0.0005	0.0004	±0.0005	0	0

Table 8 False positive rates for unknown RPC attacks (AUT09) on test data

6.3 Summary

This contribution introduces a general method that facilitates the combination of protocol analysis and payload-based anomaly detection. We present a novel data representation, so called c_k-grams, that allows to integrate protocol features and sequential features in an unified geometric feature space. Extensive experiments are conducted on recorded network traffic featuring both text-based and binary application-layer protocols which demonstrate superior accuracy on the detection of unknown attacks. The proposition shows that novel attacks can be identified reliably and the detection accuracy can be boosted from 44% using unsupervised anomaly detection with plain sequential features to 80% using combined features in the presence of attack obfuscation. Although the additional effort of protocol analysis does not seem to pay off for simple SQL injections the method proves to be especially useful for the detection of web application attacks such as XSS and SQL injections in the presence of attack obfuscation by payload customization. Moreover, this contribution shows how c_k-grams can be used to explain geometric anomalies to security experts and also provide insight into vulnerabilities by identifying and pinpointing meaningful features in a payload stream using the feature shading technique. Due to its general nature, the proposed feature extraction method can be applied on any protocol for which a protocol analyzer is available.

CHAPTER 7

Learning and classification of malware behavior

7.1 Introduction

The proliferation of malware poses a significant threat to the security of inter-connected systems. Particularly, the exponential growth of this threat driven by the diversity of existing malware and new malware families as well as the increasing sophistication of malware authors to use obfuscation and polymorphism techniques undermines state-of-the-art malware detection which predominantly relies on signature-based content-level static malware analysis. Dynamic malware analysis as an alternative, allows to monitor behavior of malware at execution time. While it is more difficult to conceal malware characteristics in presence of run-time analysis techniques, this method is also crucial to identify and understand behavioral patterns shared between individual malware instances and malware families. These insights are useful for signature updates or as an input for adjustment of heuristic rules deployed in malware detection tools.
Based on the collection of malware behavior reports, this contribution proposes a methodology for learning the behavior of malware from labeled samples and constructing behavior-based models capable of classifying unknown variants of known malware families.
Primary objective of this contribution are to investigate to what extent unknown malware (e.g. new or polymorphic versions of known malware) can be classified as instances of known malware families or as novel malware strain. Furthermore, this contribution investigates discriminative features that allow to separate between known malware families.
To this end, full-fledged cross-validation experiments are conducted using SVM-based classification over a corpus of 10.000 malware samples collected over several months using the *nephentes* honeypot platform [92]. Labels are obtained from running executing the malware samples in a secure environment against *Avira AntiVir* anti-virus solution [93]. Behavior of malware samples is recorded using the malware analysis tool *CWSandbox* [35].

7.2 Publication

Learning and Classification of Malware Behavior

Konrad Rieck[1], Thorsten Holz[2], Carsten Willems[2],
Patrick Düssel[1], and Pavel Laskov[1,3]

[1] Fraunhofer Institute FIRST
Intelligent Data Analysis Department, Berlin, Germany
[2] University of Mannheim
Laboratory for Dependable Distributed Systems, Mannheim, Germany
[3] University of Tübingen
Wilhelm-Schickard-Institute for Computer Science, Tübingen, Germany

Abstract. Malicious software in form of Internet worms, computer viruses, and Trojan horses poses a major threat to the security of networked systems. The diversity and amount of its variants severely undermine the effectiveness of classical signature-based detection. Yet variants of malware families share typical *behavioral patterns* reflecting its origin and purpose. We aim to exploit these shared patterns for classification of malware and propose a method for learning and discrimination of malware behavior. Our method proceeds in three stages: (a) behavior of collected malware is monitored in a sandbox environment, (b) based on a corpus of malware labeled by an anti-virus scanner a *malware behavior classifier* is trained using learning techniques and (c) discriminative features of the behavior models are ranked for explanation of classification decisions. Experiments with different heterogeneous test data collected over several months using honeypots demonstrate the effectiveness of our method, especially in detecting *novel* instances of malware families previously not recognized by commercial anti-virus software.

1 Introduction

Proliferation of malware poses a major threat to modern information technology. According to a recent report by Microsoft [1], every third scan for malware results in a positive detection. Security of modern computer systems thus critically depends on the ability to keep anti-malware products up-to-date and abreast of current malware developments. This has proved to be a daunting task. Malware has evolved into a powerful instrument for illegal commercial activity, and a significant effort is made by its authors to thwart detection by anti-malware products. As a result, new malware variants are discovered at an alarmingly high rate, some malware families featuring tens of thousands of currently known variants.

In order to stay alive in the arms race against malware writers, developers of anti-malware software heavily rely on automatic malware analysis tools. Unfortunately, malware analysis is obstructed by hiding techniques such as polymorphism and obfuscation. These techniques are especially effective against byte-level content analysis [17, 19] and static malware analysis methods [8, 10, 11]. In contrast to static techniques, dynamic analysis of binaries during run-time enables monitoring of malware

behavior, which is more difficult to conceal. Hence, a substantial amount of recent work has focused on development of tools for collecting, monitoring and run-time analysis of malware [3, 5, 6, 14, 22, 23, 25, 27, 36, 38].

Yet the means for collection and run-time analysis of malware by itself is not sufficient to alleviate a threat posed by novel malware. What is needed is the ability to *automatically* infer characteristics from observed malware behavior that are essential for detection and categorization of malware. Such characteristics can be used for signature updates or as an input for adjustment of heuristic rules deployed in malware detection tools. The method for automatic classification of malware behavior proposed in this contribution develops such a characterization of previously unknown malware instances by providing answers to the following questions:

1. *Does an unknown malware instance belong to a known malware family or does it constitute a novel malware strain?*
2. *What behavioral features are discriminative for distinguishing instances of one malware family from those of other families?*

We address these questions by proposing a methodology for *learning* the behavior of malware from labeled samples and constructing models capable of classifying unknown variants of known malware families while rejecting behavior of benign binaries and malware families not considered during learning. The key elements of this approach are the following:

(a) Malware binaries are collected via honeypots and spam-traps, and malware family labels are generated by running an anti-virus tool on each binary. To assess *behavioral patterns* shared by instances of the same malware family, the behavior of each binary is monitored in a sandbox environment and behavior-based analysis reports summarizing operations, such as opening an outgoing IRC connection or stopping a network service, are generated. Technical details on the collection of our malware corpus and the monitoring of malware behavior are provided in Sections 3.1–3.2.

(b) The learning algorithm in our methodology embeds the generated analysis reports in a high-dimensional vector space and learns a *discriminative model* for each malware family, i.e., a function that, being applied to behavioral patterns of an unknown malware instance, predicts whether this instance belongs to a known family or not. Combining decisions of individual discriminative models provides an answer to the first question stated above. The embedding and learning procedures are presented in Sections 3.3– 3.4.

(c) To understand the importance of specific features for classification of malware behavior, we exploit the fact that our learning model is defined by weights of behavioral patterns encountered during the learning phase. By sorting these weights and considering the most prominent patterns, we obtain characteristic features for each malware family. Details of this feature ranking are provided in Section 3.5.

We have evaluated our method on a large corpus of recent malware obtained from honeypots and spam-traps. Our results show that 70% of malware instances not identified by an anti-virus software can be correctly classified by our approach. Although such

accuracy may not seem impressive, in practice it means that the proposed method would provide correct detections in two thirds of hard cases *when anti-malware products fail*. We have also performed, as a sanity check, classification of benign executables against known malware families, and observed 100% detection accuracy. This confirms that the features learned from the training corpus are indeed characteristic for malware and not obtained by chance. The manual analysis of most prominent features produced by our discriminative models has produced insights into the relationships between known malware families. Details of experimental evaluation of our method are provided in Section 4.

2 Related work

Extensive literature exists on static analysis of malicious binaries, e.g. [8, 10, 18, 20]. While static analysis offers a significant improvement in malware detection accuracy compared to traditional pattern matching, its main weakness lies in the difficulty to handle obfuscated and self-modifying code [33]. Moreover, recent work of Moser et al. presents obfuscation techniques that are provably NP-hard for static analysis [24].

Dynamic malware analysis techniques have previously focused on obtaining reliable and accurate information on execution of malicious programs [5, 6, 23, 38]. As it was mentioned in the introduction, the main focus of our work lies in *automatic processing* of information collected from dynamic malware analysis. Two techniques for behavior-based malware analysis using clustering of behavior reports have been recently proposed [4, 21]. Both methods transform reports of observed behavior into sequences and use sequential distances (the normalized compression distance and the edit distance, respectively) to group them into clusters which are believed to correspond to malware families. The main difficulty of clustering methods stems from their unsupervised nature, i.e., the lack of any external information provided to guide analysis of data. Let us illustrate some practical problems of clustering-based approaches.

A major issue for any clustering method is to decide how many clusters are present in the data. As it is pointed out by Bailey et al. [4], there is a trade-off between cluster size and the number of clusters controlled by a parameter called *consistency* which measures a ratio between intra-cluster and inter-cluster variation. A good clustering should exhibit high consistency, i.e., uniform behavior should be observed within clusters and heterogeneous behavior between different clusters. Yet in the case of malware behavior – which is heterogeneous by its nature – this seemingly trivial observation implies that a *large* number of *small* classes is observed if consistency is to be kept high. The results in [4] yield a compelling evidence to this phenomenon: given 100% consistency, a clustering algorithm generated from a total of 3,698 malware samples 403 clusters, of which 206 (51%) contain just one single executable. What a practitioner is looking for, however, is exactly the opposite: a *small* number of *large* clusters in which variants belong to the same family. The only way to attain this effect using consistency is to play with different consistency levels, which (a) defeats the purpose of automatic classification and (b) may still be difficult to attain at a single consistency level.

Another recent approach to dynamic malware analysis is based on mining of malicious behavior reports [9]. Its main idea is to identify differences between malware

samples and benign executables, which can be used as specification of malicious behavior (malspecs). In contrast to this work, the aim of our approach is discrimination between families of malware instead of discrimination between specific malware instances and benign executables.

3 Methodology

Current malware is characterized by rich and versatile behavior, although large families of malware, such as all variants of the Allaple worm, share common behavioral patterns, e.g., acquiring and locking of particular mutexes on infected systems. We aim to exploit these shared patterns using *machine learning techniques* and propose a method capable of automatically classifying malware families based on their behavior. An outline of our learning approach is given by the following basic steps:

1. *Data acquisition.* A corpus of malware binaries currently spreading in the wild is collected using a variety of techniques, such as honeypots and spam-traps. An antivirus engine is applied to identify known malware instances and to enable learning and subsequent classification of family-specific behavior.
2. *Behavior Monitoring*. Malware binaries are executed and monitored in a sandbox environment. Based on state changes in the environment – in terms of API function calls – a behavior-based analysis report is generated.
3. *Feature Extraction*. Features reflecting behavioral patterns, such as opening a file, locking a mutex, or setting a registry key, are extracted from the analysis reports and used to embed the malware behavior into a high-dimensional vector space.
4. *Learning and Classification.* Machine learning techniques are applied for identifying the shared behavior of each malware family. Finally, a combined classifier for all families is constructed and applied to different testing data.
5. *Explanation.* The discriminative model for each malware family is analyzed using the weight vector expressing the contribution of behavioral patterns. The most prominent patterns yield insights into the classification model and reveal relations between malware families.

In the following sections we discuss these individual steps and corresponding technical background in more detail – providing examples of analysis reports, describing the vectorial representation, and explaining the applied learning algorithms.

3.1 Malware Corpus for Learning

Our malware collection used for learning and subsequent classification of malware behavior comprises more than 10,000 unique samples obtained using different collection techniques. The majority of these samples was gathered via *nepenthes*, a honeypot solution optimized for malware collection [3]. The basic principle of nepenthes is to emulate only the *vulnerable* parts of an exploitable network service: a piece of self-replicating malware spreading in the wild will be tricked into exploiting the emulated vulnerability. By automatically analyzing the received payload, we can then obtain a binary copy

of the malware itself. This leads to an effective solution for collecting self-propagating malware such as a wide variety of worms and bots. Additionally, our data corpus contains malware samples collected via *spam-traps*. We closely monitor several mailboxes and catch malware propagating via malicious e-mails, e.g., via links embedded in message bodies or attachments of e-mails. With the help of spam-traps, we are able to obtain malware such as Trojan horses and network backdoors.

The capturing procedure based on honeypots and spam-traps ensures that all samples in the corpus are *malicious*, as they were either collected while exploiting a vulnerability in a network service or contained in malicious e-mail content. Moreover, the resulting learning corpus is *current*, as all malware binaries were collected within 5 months (starting from May 2007) and reflect malware families actively spreading in the wild. In the current prototype, we focus on samples collected via honeypots and spam-traps. However, our general methodology on malware classification can be easily extended to include further malware classes, such as rootkits and other forms of non-self-propagating malware, by supplying the corpus with additional collection sources.

After collecting malware samples, we applied the anti-virus (AV) engine *Avira AntiVir* [2] to partition the corpus into common families of malware, such as variants of RBot, SDBot and Gobot. We chose Avira AntiVir as it had one of the best detection rates of 29 products in a recent AV-Test and detected 99.29% of 874,822 unique malware samples [35]. We selected the 14 malware families obtained from the most common labels assigned by Avira AntiVir on our malware corpus. These families listed in Table 1 represent a broad range of malware classes such as Trojan horses, Internet worms and bots. Note that binaries not identified by Avira AntiVir are excluded from the malware corpus. Furthermore, the contribution of each family is restricted to a maximum of 1,500 samples resulting in 10,072 unique binaries of 14 families.

Table 1. Malware families assigned by Avira AntiVir in malware corpus of 10,072 samples. The numbers in brackets indicate occurrences of each malware family in the corpus.

1: Backdoor.VanBot	(91)	8: Worm.Korgo	(244)
2: Trojan.Bancos	(279)	9: Worm.Parite	(1215)
3: Trojan.Banker	(834)	10: Worm.PoeBot	(140)
4: Worm.Allaple	(1500)	11: Worm.RBot	(1399)
5: Worm.Doomber	(426)	12: Worm.Sality	(661)
6: Worm.Gobot	(777)	13: Worm.SdBot	(777)
7: Worm.IRCBot	(229)	14: Worm.Virut	(1500)

Using an AV engine for labeling malware families introduces a problem: AV labels are generated by human analysts and are prone to errors. However, the learning method employed in our approach (Section 3.4) is well-known for its generalization ability in presence of classifcation noise [34]. Moreover, our methodology is not bound to a particular AV engine and our setup can easily be adapted to other AV engines and labels or a combination thereof.

3.2 Monitoring Malware Behavior

The behavior of malware samples in our corpus is monitored using *CWSandbox* – an analysis software generating reports of observed program operations [38]. The samples are executed for a limited time in a native Windows environment and their behavior is logged during run-time. CWSandbox implements this monitoring by using a technique called *API hooking* [13]. Based on the run-time observations, a detailed report is generated comprising, among others, the following information for each analyzed binary:

- Changes to the file system, e.g., creation, modification or deletion of files.
- Changes to the Windows registry, e.g., creation or modification of registry keys.
- Infection of running processes, e.g., to insert malicious code into other processes.
- Creation and acquiring of mutexes, e.g. for exclusive access to system resources.
- Network activity and transfer, e.g., outbound IRC connections or ping scans.
- Starting and stopping of Windows services, e.g., to stop common AV software.

Figure 1 provides examples of observed operations contained in analysis reports, e.g., copying of a file to another location or setting a registry key to a particular value. Note, that the tool provides a high-level summary of the observed events and often more than one related API call is aggregated into a single operation.

```
copy_file (filetype="File" srcfile="c:\1ae8b19ecea1b65705595b245f2971ee.exe",
  dstfile="C:\WINDOWS\system32\urdvxc.exe", flags="SECURITY_ANONYMOUS")

set_value (key="HKEY_CLASSES_ROOT\CLSID\{3534943...2312F5C0&}",
  data="lsslwhxtettntbkr")

create_process (commandline="C:\WINDOWS\system32\urdvxc.exe /start",
  targetpid="1396", showwindow="SW_HIDE", apifunction="CreateProcessA")

create_mutex (name="GhostBOT0.58b", owned="1")

connection (transportprotocol="TCP", remoteaddr="XXX.XXX.XXX.XXX",
  remoteport="27555", protocol="IRC", connectionestablished="1", socket="1780")

irc_data (username="XP-2398", hostname="XP-2398", servername="0",
  realname="ADMINISTRATOR", password="r0flc0mz", nick="[P33-DEU-51371]")
```

Fig. 1. Examples of operations as reported by CWSandbox during run-time analysis of different malware binaries. The IP address in the fifth example is sanitized.

3.3 Feature Extraction and Embedding

The analysis reports provide detailed information about malware behavior, yet raw reports are not suitable for application of learning techniques as these usually operate on vectorial data. To address this issue we derive a generic technique for mapping analysis reports to a high-dimensional feature space.

Our approach builds on the *vector space model* and *bag-of-words model*; two similar techniques previously used in the domains of information retrieval [29] and text processing [15, 16]. A document – in our case an analysis report – is characterized by frequencies of contained strings. We refer to the set of considered strings as feature set $\mathcal{F}$ and denote the set of all possible reports by $\mathcal{X}$. Given a string $s \in \mathcal{F}$ and a report $x \in \mathcal{X}$, we determine the number of occurrences of s in x and obtain the frequency $f(x, s)$. The frequency of a string s acts as a measure of its importance in x, e.g., $f(x, s) = 0$ corresponds to no importance of s, while $f(x, s) > 0.5$ indicates dominance of s in x. We derive an embedding function ϕ which maps analysis reports to an $|\mathcal{F}|$-dimensional vector space by considering the frequencies of all strings in $\mathcal{F}$:

$$\phi : \mathcal{X} \to \mathbb{R}^{|\mathcal{F}|}, \quad \phi(x) \mapsto (f(x, s))_{s \in \mathcal{F}}$$

For example, if $\mathcal{F}$ contains the strings `copy_file` and `create_mutex`, two dimensions in the resulting vector space correspond to the frequencies of these strings in analysis reports. Computation of these high-dimensional vectors seems infeasible at a first glance, as $\mathcal{F}$ may contain arbitrary many strings, yet there exist efficient algorithms that exploit the sparsity of this vector representation to achieve linear run-time complexity in the number of input bytes [28, 31].

In contrast to textual documents we can not define a feature set $\mathcal{F}$ a priori, simply because not all important strings present in reports are known in advance. Instead, we define $\mathcal{F}$ *implicitly* by deriving string features from the observed malware operations. Each monitored operation can be represented by a string containing its name and a list of key-value pairs, e.g., a simplified string s for copying a file is given by

```
"copy_file (srcfile=A, dstfile=B)"
```

Such representation yields a very specific feature set $\mathcal{F}$, so that slightly deviating behavior is reflected in different strings and vector space dimensions. Behavioral patterns of malware, however, often express variability induced by obfuscation techniques, e.g., the destination for copying a file might be a random file name. To address this problem, we represent each operation by *multiple strings* of different specificity. For each operation we obtain these strings by defining subsets of key-value pairs ranging from the full to a coarse representation. E.g. the previous example for copying a file is associated with three strings in the feature set $\mathcal{F}$

$$\texttt{"copy\_file ..."} \quad \longrightarrow \quad \begin{cases} \texttt{"copy\_file\_1 (srcfile=A, dstfile=B)"} \\ \texttt{"copy\_file\_2 (srcfile=A)"} \\ \texttt{"copy\_file\_3 ()"} \end{cases}$$

The resulting implicit feature set $\mathcal{F}$ and the vector space induced by ϕ correspond to various strings of possible operations, values and attributes, thus covering a wide range of potential malware behavior. Note, that the embedding of analysis reports using a feature set $\mathcal{F}$ and function ϕ is generic, so that it can be easily adapted to different report formats of malware analysis software.

3.4 Learning and Classification

The embedding function ϕ introduced in the previous section maps analysis reports into a vector space in which various learning algorithms can be applied. We use the well-established method of *Support Vector Machines* (SVM), which provides strong generalization even in presence of noise in features and labels. Given data of two classes an SVM determines an *optimal hyperplane* that separates points from both classes with maximal margin [e.g. 7, 30, 34].

The optimal hyperplane is represented by a vector w and a scalar b such that the inner product of w with vectors $\phi(x_i)$ of the two classes are separated by an interval between -1 and $+1$ subject to b:

$$\langle w, \phi(x_i)\rangle + b \geq +1, \text{ for } x_i \text{ in class 1,}$$
$$\langle w, \phi(x_i)\rangle + b \leq -1, \text{ for } x_i \text{ in class 2.}$$

The optimization problem to be solved for finding w and b can be solely formulated in terms of inner products $\langle \phi(x_i), \phi(x_j)\rangle$ between data points. In practice these inner products are computed by so called *kernel functions*, which lead to non-linear classification surfaces. For example, the kernel function k for polynomials of degree d used in our experiments is given by

$$k(x_i, x_j) = (\langle \phi(x_i), \phi(x_j)\rangle + 1)^d.$$

Once trained, an SVM classifies a new report x by computing its distance $h(x)$ from the separating hyperplane as

$$h(x) = \langle w, \phi(x)\rangle + b = \sum_{i=1}^{n} \alpha_i y_i k(x_i, x) + b,$$

where α_i are parameters obtained during training and y_i labels ($+1$ or -1) of training data points. The distance $h(x)$ can then be used for multi-class classification among malware families in one of the following ways:

1. *Maximum distance.* A label is assigned to a new behavior report by choosing the classifier with the highest positive score, reflecting the distance to the most discriminative hyperplane.
2. *Maximum probability estimate.* Additional calibration of the outputs of SVM classifiers allows to interpret them as probability estimates. Under some mild probabilistic assumptions, the conditional posterior probability of the class $+1$ can be expressed as:
$$P(y = +1 \,|\, h(x)) = \frac{1}{1 + \exp(Ah(x) + B)},$$
where the parameters A and B are estimated by a logistic regression fit on an independent training data set [26]. Using these probability estimates, we choose the malware family with the highest estimate as our classification result.

In the following experiments we will use the maximum distance approach for combining the output of individual SVM classifiers. The probabilistic approach is applicable to prediction as well as detection of novel malware behavior and will be considered in Section 4.3.

3.5 Explanation of Classification

A security practitioner is not only interested in how accurate a learning system performs, but also needs to understand how such performance is achieved – a requirement not satisfied by many "black-box" applications of machine learning. In this section we supplement our proposed methodology and provide a procedure for explaining classification results obtained using our method.

The discriminative model for classification of a malware family is the hyperplane w in the vector space $\mathbb{R}^{|\mathcal{F}|}$ learned by an SVM. As the underlying feature set $\mathcal{F}$ corresponds to strings $s_i \in \mathcal{F}$ reflecting observed malware operations, each dimension w_i of w expresses the contribution of an operation to the decision function h. Dimensions w_i with high values indicate strong discriminative influence, while dimensions with low values express few impact on the decision function. By sorting the components w_i of w one obtains a *feature ranking*, such that $w_i > w_j$ implies higher relevance of s_i over s_j. The most prominent strings associated with the highest components of w can be used to gain insights into the trained decision function and represent typical behavioral patterns of the corresponding malware family.

Please note that an explicit representation of w is required for computing a feature ranking, so that in the following we provide explanations of learned models only for polynomial kernel functions of degree 1.

4 Experiments

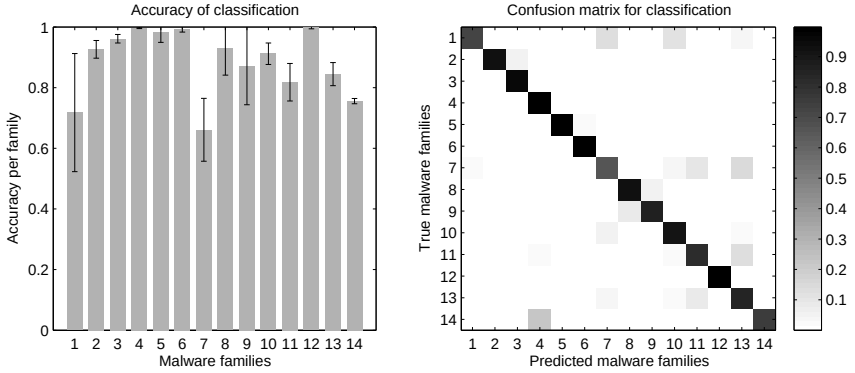
We now proceed to evaluate the performance and effectiveness of our methodology in different setups. For all experiments we pursue the following experimental procedure: The malware corpus of 10,072 samples introduced in Section 3.1 is randomly split into three partitions, a *training*, *validation* and *testing* partition. For each partition behavior-based reports are generated and transformed into a vectorial representation as discussed in Section 3. The training partition is used to learn individual SVM classifiers for each of the 14 malware families using different parameters for regularization and kernel functions. The best classifier for each malware family is then selected using the classification accuracy obtained on the validation partition. Finally, the overall performance is measured using the combined classifier on the testing partition.

This procedure, including randomly partitioning the malware corpus, is repeated over five experimental runs and corresponding results are averaged. For experiments involving data not contained in the malware corpus (Section 4.2 and 4.3), the testing partition is replaced with malware binaries from a different source. The machine learning toolbox *Shogun* [32] has been chosen as an implementation of the SVM. The toolbox has been designed for large-scale experiments and enables learning and classification of 1,700 samples per minute and malware family.

4.1 Classification of Malware Behavior

In the first experiment we examine the general classification performance of our malware behavior classifier. Testing data is taken from the malware corpus introduced in

Section 3.1. In Figure 2 the per-family accuracy and a confusion matrix for this experiment is shown. The plot in Figure 2 (a) depicts the percentage of correctly assigned labels for each of the 14 selected malware families. Error bars indicate the variance measured during the experimental runs. The matrix in Figure 2 (b) illustrates confusions made by the malware behavior classifier. The density of each cell gives the percentage of a true malware family assigned to a predicted family by the classifier. The matrix diagonal corresponds to correct classification assignments.



(a) Accuracy per malware family (b) Confusion of malware families

Fig. 2. Performance of malware behavior classifier using operation features on testing partition of malware corpus. Results are averaged over five experimental runs.

On average 88% of the provided testing binaries are correctly assigned to malware families. In particular, the malware families Worm.Allaple (4), Worm.Doomber (5), Worm.Gobot (6) and Worm.Sality (12) are identified almost perfectly. The precise classification of Worm.Allaple demonstrates the potential of our methodology, as this type of malware is hard to detect using static methods: Allaple is polymorphically encrypted, i.e., every copy of the worm is different from each other. This means that static analysis can only rely on small parts of the malware samples, e.g., try to detect the decryptor. However, when the binary is started, it goes through the polymorphic decryptor, unpacks itself, and then proceeds to the static part of the code, which we observe with our methodology. All samples express a set of shared behavioral patterns sufficient for classification using our behavior-based learning approach.

The accuracy for Backdoor.VanBot (1) and Worm.IRCBot (7) reaches around 60% and expresses larger variance – an indication for a generic AV label characterizing multiple malware strains. In fact, the samples of Worm.IRCBot (7) in our corpus comprise over 80 different mutex names, such as `SyMMeC`, `itcrew` or `h1dd3n`, giving evidence of the heterogeneous labeling.

4.2 Prediction of Malware Families

In order to evaluate how good we can even *predict* malware families which are not detected by anti-virus products, we extended our first experiment. As outlined in Section 3.1, our malware corpus is generated by collecting malware samples with the help of honeypots and spam-traps. The anti-virus engine Avira AntiVir, used to assign labels to the 10,072 binaries in our malware corpus, failed to identify additional 8,082 collected malware binaries. At this point, however, we can not immediately assess the performance of our malware behavior classifier as the *ground truth*, the true malware families of these 8,082 binaries, is unknown.

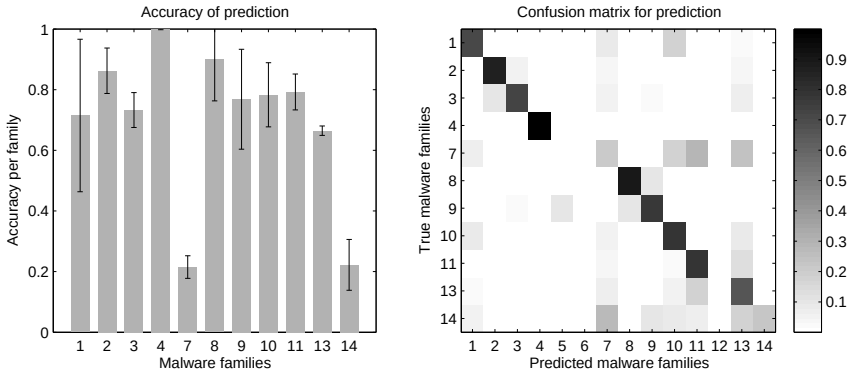
We resolve this problem by re-scanning the undetected binaries with the Avira AntiVir engine after a period of four weeks. The rationale behind this approach is that the AV vendor had time to generate and add missing signatures for the malware binaries and thus several previously undetected samples could be identified. From the total of 8,082 undetected binaries, we now obtain labels for 3,139 samples belonging to the 14 selected malware families. Table 2 lists the number of binaries for each of the 14 families. Samples for Worm.Doomber, Worm.Gobot and Worm.Sality were not present, probably because these malware families did not evolve and current signatures were sufficient for accurate detection.

Table 2. Undetected malware families of 3,139 samples, labeled by Avira AntiVir four weeks after learning phase. Numbers in brackets indicate occurrences of each Malware family.

1: Backdoor.VanBot	(169)	8: Worm.Korgo	(4)
2: Trojan.Bancos	(208)	9: Worm.Parite	(19)
3: Trojan.Banker	(185)	10: Worm.PoeBot	(188)
4: Worm.Allaple	(614)	11: Worm.RBot	(904)
5: Worm.Doomber	(0)	12: Worm.Sality	(0)
6: Worm.Gobot	(0)	13: Worm.SdBot	(597)
7: Worm.IRCBot	(107)	14: Worm.Virut	(144)

Based on the experimental procedure used in the first experiment, we replace the original testing data with the embedded behavior-based reports of the new 3,139 labeled samples and again perform five experimental runs.

Figure 3 provides the per-family accuracy and the confusion matrix achieved on the 3,139 malware samples. The overall result of this experiment is twofold. On average, 69% of the malware behavior is classified correctly. Some malware, most notably Worm.Allaple (4), is detected with high accuracy, while on the other hand malware families such as Worm.IRCBot (7) and Worm.Virut (14) are poorly recognized. Still, the performance of our malware behavior classifier is promising, provided that during the learning phase *none* of these malware samples was detected by the Avira AntiVir engine. Moreover, the fact that AV signatures present during learning did not suffice for detecting these binaries might also indicate truly novel behavior of malware, which is impossible to predict using behavioral patterns contained in our malware corpus.



(a) Accuracy per malware family (b) Confusion of malware families

Fig. 3. Performance of malware behavior classifier on undetected data using operation features. Malware families 5, 6 and 12 are not present in the testing data.

4.3 Identification of Unknown Behavior

In the previous experiments we considered the performance of our malware behavior classifier on 14 fixed malware families. In a general setting, however, a classifier might also be exposed to malware binaries that do *not* belong to one of these 14 families. Even if the majority of current malware families would be included in a large learning system, future malware families could express activity not matching any patterns of previously monitored behavior. Moreover, a malware behavior classifier might also be exposed to benign binaries either by accident or in terms of a denial-of-service attack. Hence, it is crucial for such a classifier to not only identify particular malware families with high accuracy, but also to verify the confidence of its decision and report unknown behavior.

We extend our behavior classifier to identify and reject *unknown behavior* by changing the way individual SVM classifiers are combined. Instead of using the maximum distance to determine the current family, we consider probability estimates for each family as discussed in Section 3.4. Given a malware sample, we now require *exactly one* SVM classifier to yield a probability estimate larger 50% and *reject* all other cases as unknown behavior.

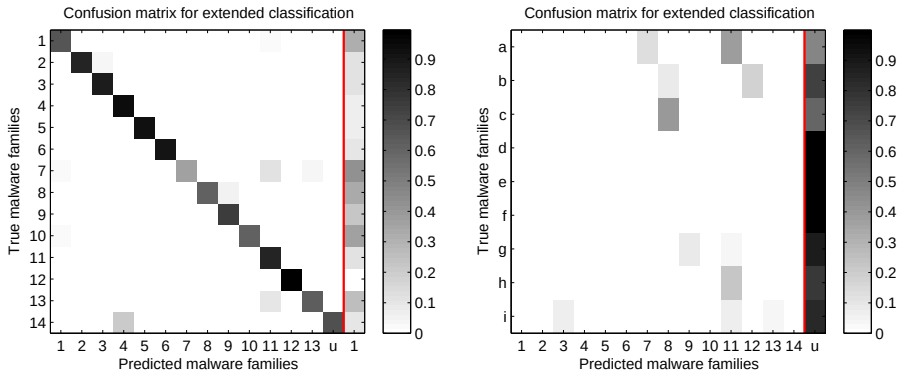
For evaluation of this extended behavior classifier we consider additional malware families not part of our malware corpus and benign binaries randomly chosen from several desktop workstations running Windows XP SP2. Table 3 provides an overview of the additional malware families. We perform three experiments: first, we repeat the experiment of Section 4.1 with the extended classifier capable of rejecting unknown behavior, second we consider 530 samples of the unknown malware families given in Table 3 and third we provide 498 benign binaries to the extended classifier.

Figure 4 shows results of the first two experiments averaged over five individual runs. The confusion matrices in both sub-figures are extended by a column labeled *u* which contains the percentage of predicted unknown behavior. Figure 4 (a) depicts the confusion matrix for the extended behavior classifier on testing data used in Sec-

Table 3. Malware families of 530 samples not contained in malware learning corpus. The numbers in brackets indicate occurrences of each malware family.

a: Worm.Spybot	(63)	f: Trojan.Proxy.Cimuz	(73)
b: Worm.Sasser	(23)	g: Backdoor.Zapchast	(25)
c: Worm.Padobot	(62)	h: Backdoor.Prorat	(77)
d: Worm.Bagle	(20)	i: Backdoor.Hupigon	(96)
e: Trojan.Proxy.Horst	(29)		

tion 4.1. In comparison to Section 4.1 the overall accuracy decreases from 88% to 76%, as some malware behavior is classified as unknown, e.g., for the generic AV labels of Worm.IRCBot (7). Yet this increase in false-positives coincides with decreasing confusions among malware families, so that the confusion matrix in Figure 4 (a) yields fewer off-diagonal elements in comparison to Figure 2 (b). Hence, the result of using a probabilistic combination of SVM classifiers is twofold: on the one hand behavior of some malware samples is indicated as unknown, while on the other hand the amount of confusions is reduced leading to classification results supported by strong confidence.



(a) Confusion on testing data (b) Confusion on unknown malware

Fig. 4. Performance of extended behavior classifier on (a) original testing data and (b) malware families not contained in learning corpus. The column labeled "u" corresponds to malware binaries classified as *unknown behavior*.

Figure 4 (b) now provides the confusion matrix for the unknown malware families given in Table 3. For several of these families no confusion occurs at all, e.g., for Worm.Bagle (d), Trojan.Proxy.Horst (e) and Trojan.Proxy.Cimuz (f). The malware behavior classifier precisely recognizes that these binaries do not belong to one of the 14 malware families used in our previous experiments. The other tested unknown malware families show little confusion with one of the learned families, yet the majority of these

confusions can be explained and emphasizes the capability of our methodology to not discriminate AV labels of malware but its behavior.

- Worm.Spybot (a) is similar to other IRC-bots in that it uses IRC as command infrastructure. Moreover, it exploits vulnerabilities in network services and creates auto-start keys to enable automatic start-up after system reboot. This behavior leads to confusion with Worm.IRCBot (7) and Worm.RBot (11), which behave in exactly the same way.
- Worm.Padobot (c) is a synonym for Worm.Korgo (8): several AV engines name this malware family Worm.Padobot, whereas others denote it by Worm.Korgo. The corresponding confusion in Figure 4 (b) thus results from the ability of our learning method to generalize beyond the restricted set of provided labels.
- Backdoor.Zapchast (g) is a network backdoor controlled via IRC. Some binaries contained in variants of this malware are infected with Worm.Parite (9). This coupling of two different malware families, whether intentional by the malware author or accidental, is precisely reflected in a small amount of confusion shown in Figure 4 (b).

In the third experiment focusing on benign binaries, all reports of benign behavior are correctly assigned to the unknown class and rejected by the extended classifier. This result shows that the proposed learning method captures typical behavioral patterns of malware, which leads to few confusions with other malware families but enables accurate discrimination of normal program behavior if provided as input to a classifier.

4.4 Explaining Malware Behavior Classification

The experiments in the previous sections demonstrate the ability of machine learning techniques to effectively discriminate malware behavior. In this section we examine the discriminative models learned by the SVM classifiers and show that relations of malware beyond the provided AV labels can be deduced from the learned classifiers. For each of the 14 considered malware families we learn an SVM classifier, such that there exist 14 hyperplanes separating the behavior of one malware family from all others. We present the learned decision functions for the Sality and Doomber classifiers as outlined in Section 3.5 by considering the most prominent patterns in their weight vectors.

Sality Classifier Figure 5 depicts the top five discriminating operation features for the family Worm.Sality learned by our classifier. Based on this example, we see that operation features can be used by a human analyst to understand the actual behavior of the malware family, e.g., the first two features show that Sality creates a file within the Windows system directory. Since both variants created during the preprocessing step (see Section 3.3 for details) are included, this indicates that Sality commonly uses the source filename `vcmgcd32.dl_`. Moreover, this malware family also deletes at least one file within the Windows system directory. Furthermore, this family creates a mutex containing the string `kuku_joker` (e.g., `kuku_joker_v3.09` as shown in Figure 5 and

```
0.0142: create_file_2 (srcpath="C:\windows\...")
0.0073: create_file_1 (srcpath="C:\windows\...", srcfile="vcmgcd32.dl_")
0.0068: delete_file_2 (srcpath="C:\windows\...")
0.0051: create_mutex_1 (name="kuku_joker_v3.09")
0.0035: enum_processes_1 (apifunction="Process32First")
```

Fig. 5. Discriminative operation features extracted from the SVM classifier of the the malware family *Sality*. The numbers to the left are the sorted components of the hyperplane vector *w*.

```
0.0084: create_mutex_1 (name="GhostBOT0.58c")
0.0073: create_mutex_1 (name="GhostBOT0.58b")
0.0052: create_mutex_1 (name="GhostBOT0.58a")
0.0014: enum_processes_1 (apifunction="Process32First")
0.0011: query_value_2 (key="HKEY_LOCAL...\run", subkey_or_value="GUARD")
```

Fig. 6. Discriminative operation features extracted from the SVM classifier of the the malware family *Doomber*. The numbers to the left are the sorted components of the hyperplane vector *w*.

`kuku_joker_v3.04` as sixth most significant feature) such that only one instance of the binary is executed at a time. Last, Sality commonly enumerates the running processes.

Based on these operation features, we get an overview of what specific behavior is characteristic for a given malware family; we can *understand* what the behavioral patterns for one family are and how a learned classifier operates.

Doomber Classifier In Figure 6, we depict the top five discriminating operation features for Worm.Doomber. Different features are significant for Doomber compared to Sality: the three most significant components for this family are similar mutex names, indicating different versions contained in our malware corpus. Furthermore, we can see that Doomber enumerates the running processes and queries certain registry keys.

In addition, we make another interesting observation: our learning-based system identified the mutex names `GhostBOT-0.57a`, `GhostBOT-0.57` and `GhostBOT` to be among the top five operation features for Worm.Gobot. The increased version number reveals that Gobot and Doomber are closely related. Furthermore, our system identified several characteristic, additional features contained in reports from both malware families, e.g., registry keys accessed and modified by both of them. We manually verified that both families are closely related and that Doomber is indeed an enhanced version of Gobot. This illustrates that our system may also help to identify *relations* between different malware families based on observed run-time behavior.

5 Limitations

In this section, we examine the limitations of our learning and classification methodology. In particular, we discuss the drawbacks of our analysis setup and examine evasion techniques.

One drawback of our current approach is that we rely on one single program execution of a malware binary: we start the binary within the sandbox environment and observe one execution path of the sample, which is stopped either if a timeout is reached or if the malware exits from the run by itself. We thus do not get a full overview of what the binary intends to do, e.g., we could miss certain actions that are only executed on a particular date. However, this deficit can be addressed using a technique called *multi-path execution*, recently introduced by Moser et al. [23], which essentially tracks input to a running binary and selects a feasible subset of possible execution paths. Moreover, our results indicate that a single program execution often contains enough information for accurate classification of malware behavior, as malware commonly tries to aggressively propagate further or quickly contacts a Command & Control servers.

Another drawback of our methodology is potential evasion by a malware, either by detecting the existence of a sandbox environment or via mimicry of different behavior. However, detecting of the analysis environment is no general limitation of our approach: to mitigate this risk, we can easily substitute our analysis platform with a more resilient platform or even use several different analysis platforms to generate the behavior-based report. Second, a malware binary might try to mimic the behavior of a different malware family or even benign binaries, e.g. using methods proposed in [12, 37]. The considered analysis reports, however, differ from sequential representations such as system call traces in that multiple occurrences of identical activities are discarded. Thus, mimicry attacks can not arbitrarily blend the frequencies or order of operation features, so that only very little activity may be covered in a single mimicry attack.

A further weakness of the proposed supervised classification approach is its inability to find structure in new malware families not present in a training corpus. The presence of unknown malware families can be detected by the rejection mechanism used in our classifiers, yet no further distinction among rejected instances is possible. Whether this is a serious disadvantage in comparison to clustering methods is to be seen in practice.

6 Conclusions

The main contribution of this paper is a learning-based approach to automatic classification of malware behavior. The key ideas of our approach are: (a) the incorporation of labels assigned by anti-virus software to define classes for building discriminative models; (b) the use of string features describing specific behavioral patterns of malware; (c) automatic construction of discriminative models using learning algrithms and (d) identification of explanatory features of learned models by ranking behavioral patterns according to their weights. To apply our method in practice, it suffices to collect a large number of malware samples, analyse its behavior using a sandbox environment, identify typical malware families to be classified by running a standard anti-virus software and construct a malware behavior classifier by learning single-family models using a machine learning toolbox.

As a proof of concept, we have evaluated our method by analyzing a training corpus collected from honeypots and spam-traps. The set of known families consisted of 14 common malware families; 9 additional families were used to test the ability of our method to identify behavior of unknown families. In an experiment with over

3,000 previously *undetected* malware binaries, our system correctly predicted almost 70% of labels assigned by an anti-virus scanner *four weeks later*. Our method also detects unknown behavior, so that malware families not present in the learning corpus are correctly identified as unknown. The analysis of prominent features inferred by our discriminative models has shown interesting similarities between malware families; in particular, we have discovered that Doomber and Gobot worms derive from the same origin, with Doomber being an extension of Gobot.

Despite certain limitations of our current method, such as single-path execution in a sandbox and the use of imperfect labels from an anti-virus software, the proposed learning-based approach offers the possibility for accurate automatic analysis of malware behavior, which should help developers of anti-malware software to keep apace with the rapid evolution of malware.

Bibliography

[1] Microsoft Security Intelligence Report, October 2007. http://www.microsoft.com/downloads/details.aspx?FamilyID=4EDE2572-1D39-46EA-94C6-4851750A2CB0.

[2] Avira. AntiVir PersonalEdition Classic, 2007. http://www.avira.de/en/products/personal.html.

[3] P. Baecher, M. Koetter, T. Holz, M. Dornseif, and F. C. Freiling. The nepenthes platform: An efficient approach to collect malware. In *Proceedings of the 9th Symposium on Recent Advances in Intrusion Detection (RAID'06)*, pages 165–184, 2006.

[4] M. Bailey, J. Oberheide, J. Andersen, Z. M. Mao, F. Jahanian, and J. Nazario. Automated classification and analysis of internet malware. In *Proceedings of the 10th Symposium on Recent Advances in Intrusion Detection (RAID'07)*, pages 178–197, 2007.

[5] U. Bayer, C. Kruegel, and E. Kirda. TTAnalyze: A tool for analyzing malware. In *Proceedings of EICAR 2006*, April 2006.

[6] U. Bayer, A. Moser, C. Kruegel, and E. Kirda. Dynamic analysis of malicious code. *Journal in Computer Virology*, 2:67–77, 2006.

[7] C. Burges. A tutorial on support vector machines for pattern recognition. *Knowledge Discovery and Data Mining*, 2(2):121–167, 1998.

[8] M. Christodorescu and S. Jha. Static analysis of executables to detect malicious patterns. In *Proceedings of the 12th USENIX Security Symposium*, pages 12–12, 2003.

[9] M. Christodorescu, S. Jha, and C. Kruegel. Mining specifications of malicious behavior. In *Proceedings of the 6th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE)*, 2007.

[10] M. Christodorescu, S. Jha, S. A. Seshia, D. X. Song, and R. E. Bryant. Semantics-aware malware detection. In *IEEE Symposium on Security and Privacy*, pages 32–46, 2005.

[11] H. Flake. Structural comparison of executable objects. In *Proceedings of Detection of Intrusions and Malware & Vulnerability Assessment (DIMVA'04)*, 2004.

[12] P. Fogla, M. Sharif, R. Perdisci, O. Kolesnikov, and W. Lee. Polymorphic blending attacks. In *Proceedings of the 15th USENIX Security Symposium*, pages 241–256, 2006.

[13] G. C. Hunt and D. Brubacker. Detours: Binary interception of Win32 functions. In *Proceedings of the 3rd USENIX Windows NT Symposium*, pages 135–143, 1999.

[14] X. Jiang and D. Xu. Collapsar: A VM-based architecture for network attack detention center. In *Proceedings of the 13th USENIX Security Symposium*, 2004.

[15] T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *Proceedings of the European Conference on Machine Learning*, pages 137 – 142. Springer, 1998.

[16] T. Joachims. *Learning to Classify Text using Support Vector Machines*. Kluwer Academic Publishers, 2002.

[17] M. Karim, A. Walenstein, A. Lakhotia, and P. Laxmi. Malware phylogeny generation using permutations of code. *Journal in Computer Virology*, 1(1–2):13–23, 2005.

[18] E. Kirda, C. Kruegel, G. Banks, G. Vigna, and R. A. Kemmerer. Behavior-based spyware detection. In *Proceedings of the 15th USENIX Security Symposium*, pages 19–19, 2006.

[19] J. Kolter and M. Maloof. Learning to detect and classify malicious executables in the wild. *Journal of Machine Learning Research*, 7(Dec):2721 – 2744, 2006.

[20] C. Kruegel, W. Robertson, and G. Vigna. Detecting kernel-level rootkits through binary analysis. In *Proceedings of the 20th Annual Computer Security Applications Conference (ACSAC)*, 2004.

[21] T. Lee and J. J. Mody. Behavioral classification. In *Proceedings of EICAR 2006*, April 2006.

[22] C. Leita, M. Dacier, and F. Massicotte. Automatic handling of protocol dependencies and reaction to 0-day attacks with ScriptGen based honeypots. In *Proceedings of the 9th Symposium on Recent Advances in Intrusion Detection (RAID'06)*, Sep 2006.

[23] A. Moser, C. Kruegel, and E. Kirda. Exploring multiple execution paths for malware analysis. In *Proceedings of 2007 IEEE Symposium on Security and Privacy*, 2007.

[24] A. Moser, C. Kruegel, and E. Kirda. Limits of static analysis for malware detection. In *Proceedings of the 23rd Annual Computer Security Applications Conference (ACSAC)*, 2007. to appear.

[25] Norman. Norman sandbox information center. Internet: `http://sandbox.norman.no/`, Accessed: 2007.

[26] J. Platt. Probabilistic outputs for Support Vector Machines and comparison to regularized likelihood methods. In A. Smola, P. Bartlett, B. Schölkopf, and D. Schuurmans, editors, *Advances in Large Margin Classifiers*. MIT Press, 2001.

[27] F. Pouget, M. Dacier, and V. H. Pham. Leurre.com: on the advantages of deploying a large scale distributed honeypot platform. In *ECCE'05, E-Crime and Computer Conference, 29-30th March 2005, Monaco*, Mar 2005.

[28] K. Rieck and P. Laskov. Linear-time computation of similarity measures for sequential data. *Journal of Machine Learning Research*, 9(Jan):23–48, 2008.

[29] G. Salton, A. Wong, and C. Yang. A vector space model for automatic indexing. *Communications of the ACM*, 18(11):613–620, 1975.

[30] B. Schölkopf and A. Smola. *Learning with Kernels*. MIT Press, Cambridge, MA, 2002.

[31] J. Shawe-Taylor and N. Cristianini. *Kernel Methods for Pattern Analysis*. Cambridge University Press, 2004.

[32] S. Sonnenburg, G. Rätsch, C. Schäfer, and B. Schölkopf. Large scale multiple kernel learning. *Journal of Maching Learning Research*, 7:1531–1565, 2006.

[33] P. Szor. *The Art of Computer Virus Research and Defense*. Addison-Wesley, 2005.

[34] V. Vapnik. *Statistical Learning Theory*. John Wiley & Sons, 1998.

[35] Virus Bulletin. AVK tops latest AV-Test charts, August 2007. http://www.virusbtn.com/news/2007/08_22a.xml.

[36] M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. C. Snoeren, G. M. Voelker, and S. Savage. Scalability, fidelity, and containment in the potemkin virtual honeyfarm. *SIGOPS Oper. Syst. Rev.*, 39(5):148–162, 2005.

[37] D. Wagner and P. Soto. Mimicry attacks on host based intrusion detection systems. In *Proceedings of the 9th ACM Conference on Computer and Communications Security (CCS'02)*, pages 255–264, 2002.

[38] C. Willems, T. Holz, and F. Freiling. CWSandbox: Towards automated dynamic binary analysis. *IEEE Security and Privacy*, 5(2), 2007.

7.3 Summary

This contribution proposed a novel method for automatic classification of malware behavior utilizing SVM-based classification of labeled malware samples which are represented by string features describing specific behavior patterns of malware obtained from the *CWSandbox* malware analysis tool. To this end, a corpus of 10.000 malware samples was collected, their behavior analysed in the sandbox environment, and single-family behavior classifiers learned based on typical malware families identified by running a standard anti-virus software. The set of known families consisted of 14 common malware families – 9 additional families were used to test the ability of the method to identify behavior of unknown families.

Experimental results confirm the two objectives of this contribution. In an experiment with over 3.000 previously undetected malware samples, the proposed method correctly predicted approx. 70% of labels assigned by the anti-virus solution four weeks later. Furthermore, the method also reliably identified unknown behavior such that malware families not learned during training are correctly classified as "unknown". An analysis of discriminative features revealed interesting similarities between *Doomber* and *Gobot* worms, suggesting that *Doomber* being an extension of *Gobot*.

CHAPTER 8

Conclusions

Modern society relies on information technology which is exposed to a plethora of cyber threats. Increasing technology complexity and fast-paced adoption and integration of emerging technology, such as Internet of Things (IoT), increases the demand for security and resilience. As the level of attack sophistication has grown significantly, the detection of unknown cyber threats has emerged to a major challenge for security practitioners in many different domains. The detection of so called "zero-day" attacks – the exploitation of vulnerabilities unknown to the security community such that no fix or patch is available – becomes paramount for the effective protection of critical information technology infrastructures.
A number of representative studies has been presented as part of this dissertation which demonstrate the effectiveness and usefulness of machine learning methods as alternative to state-of-the-art detection approaches to solve existing unknown threat detection problems in both the network and computer security domain. Thereby, the main focal point of this dissertation is the development of machine learning methods to detect unknown cyber attacks in computer networks and furthermore derive useful information to pinpoint vulnerabilities in applications based on the analysis of network packet payloads. To this end, the studies presented make the following key contributions:

- Conducted comprehensive experiments to investigate strengths and limitations of supervised and unsupervised machine learning methods for the detection of known and unknown network attacks based on network packet header features.
- Developed and investigated the effectiveness of an anomaly detection method to detect network attacks based on the extraction and selection of language model features from network packet payload.
- Developed and investigated methods to bridge the gap between network protocol analysis and payload-based anomaly detection and allow to reflect both structural and sequential similarity aspects in learned data models.
- Investigated the effectiveness of proposed methods to other unknown threat detection challenges outside of network security, i.e. unknown malware detectionn, to demonstrate applicability of methods in other fields.

For each of the key contributions major results and conclusions are outlined below.
Numerous machine learning methods can be used to detect network attacks. In order to address the question to what extent unsupervised machine learning can be leverage as an effective alternative to

supervised learning, Chapter 2 has provided a comparative analysis of various supervised and unsupervised methods to investigate strengths and limitations in terms of capabilities and accuracy to detect unknown attacks in the presence and absence of label information. A comprehensive experimental evaluation over the well known KDD Cup 1999 data set has shown that, although supervised learning significantly outperforms unsupervised learning in the known attack scenario (i.e. attack instances were used during training/validation of supervised methods), performance of supervised learning methods dropped significantly to the level of unsupervised learning methods. In the unknown attack scenario, the Support Vector Machine attained highest detection accuracy with approx. 82% true positive at 10% false positive rate. In the unknown attack scenario, the graph-based γ-outlier detection method, which defines the distance of a data point to its k nearest neighbors as anomaly score, outperforms all other tested unsupervised methods and compares to the best supervised method (SVM) with a detection accuracy of 79% at 10% false positive rate.
From the experimental evaluation it can concluded that unsupervised machine learning can be used as an alternative to supervised methods to effectively detect unknown network intrusions as a label-free alternative to supervised learning based on network packet header features.

Chapter 3 presented an experimental evaluation of anomaly detection over both plain-text (i.e. HTTP) and binary (i.e. RPC) network packet payloads commonly seen in SCADA network traffic of industrial automation networks. The contribution proposed a fast and effective centroid-based anomaly detection method which utilizes higher order language models in combination with distance measures to detect payload-based "zero-day" attacks in SCADA network traffic on the TCP/IP layer. With a detection rate of 88-92% true positives at 0.2% false positives the method has proved to be useful for the detection of unknown attacks in SCADA network traffic. Moreover, for plain-text network protocols, the proposed method significantly outperformed the state-of-the-art anomaly detection system *PAYL* which attained 65% true positive at 0.2% false positive rate. However, the accuracy improvement of the proposed method over binary network protocols is only marginal compared to *PAYL*.
From the experimental evaluation it can be concluded that the proposed payload-based anomaly detection method using language model features extracted from network packet payloads allow for the effective detection of network attacks carried out over both plain-text and binary network protocols.

Given the plethora of available feature sets to represent data objects, selecting the right set of features to learn data models is a commonly known challenge in machine learning. Particularly, higher order language models may result in high-volume and sparse feature spaces which can impact the quality of a predictive model. Hence, the choice of the right set of features is crucial not only for learning the right model but also to allow for efficient learning and decision making. Therefore, Chapter 4 introduced a novel automated feature selection method for anomaly detection which allows for the selection and combination of an optimal set of discriminative features taken from different feature sets to realize a minimal-volume data description. Experiments showed that the proposed method outperforms any baseline using a single set of features for more than five samples to train the model of normality. The proposed method turned out to be particularly effective on small training set sizes. Furthermore, experimental results suggested that feature weighting obtained from solving a semi-infinite linear program depends on the size of the training data. While for smaller sized training sets the feature weight distribution tends to be uniform, the weight distribution is increasingly dominated by the sequential expert feature type as the training set size grows.
Based on the experimental evaluation it can be concluded that the proposed method allows to effectively identify most informative, discriminative features from different sets of feature types which helps

security practitioners to understand decision making by the predictive model and pinpoint suspicious characteristics of attack instances.

Chapter 5 introduced a SVM-based anomaly detection method that allows for the detection of unknown network attacks based on the novel *attributed token kernel* – a similarity measures for composite data objects.The kernel allows to calculate similarity between composite data structures (e.g. HTTP requests) by comparing sequential features associated with network protocol tokens obtained from parsing network packet payloads. The experimental evaluation was carried out over two different HTTP data sets recorded on publicly accessible web servers comparing the novel attributed token kernel against conventional spectrum kernels. Two realistic attack data sets were created which contain overflow-type of attacks as well as common web application attacks (e.g. SQL injection). Attack instances were merged with test data partitions during full-fledged cross validation. Experimental results revealed, that the attributed token kernel compares to the best spectrum kernel (k=1, binary feature embedding) attaining an accuracy of 99% true positive rate and less than 1% false positive rate at the detection of unknown overflow-type of attacks. This can be explained by the fact, that overflow-based attacks are not difficult to detect, generally due to the inclusion of large network packet payloads and vast utilization of non-printable characters. However, the experimental evaluation showed that for the detection of web application attacks the attributed token kernel significantly outperforms the spectrum kernel and boosts the detection rate from 70% to 100% at less than 0.14% false positives. This can be explained by the increased sensitivity of local outliers induced by protocol tokenization. Surprisingly, the proposed method also outperforms both misuse-based detection as well as the service-specific anomaly detection approach. However, a major drawback of this approach is the complexity of similarity calculation due to normalization of component-level kernel values.
Based on the experimental results it can concluded that the attributed token kernel is able to capture sequential dissimilarity with respect to specific protocol context and thus, is particularly useful to detect web application attacks which are hard to detect using kernels over language model features (e.g. Spectrum kernel).

A continuative study presented in Chapter 6 proposed a novel data representation, so called c_k-grams, which allows to integrate protocol features and sequential features extracted from a composite data object in a unified geometric feature space. The main difference to earlier work presented in Chapter 5 is that c_k-grams – as opposed to the *attributed token kernel* – reduce complexity of similarity calculation based on linear kernel functions and provide transparency of feature relevance while preserving the same information. Extensive experiments were conducted on recorded plain-text and binary application-layer network traffic. Results demonstrated that novel attacks can be identified reliably and the detection accuracy can be boosted from 44% using unsupervised anomaly detection with language model features to 80% using combined features in the presence of attack obfuscation. The method has proved to be particularly useful for the detection of web application attacks such as XSS and SQL injections in presence of attack obfuscation. Furthermore, this contribution showed how c_k-grams can be used to explain geometric anomalies to security experts and also provide insight into vulnerabilities by identifying and pinpointing meaningful features in a payload stream using the feature shading technique.
Based on the experimental evaluation it can be concluded that c_k-gram are a powerful representation for composite data objects (e.g. application-level network packet payload) that allows to significantly improve detection accuracy compared to conventional language models while reducing complexity of similarity calculation. Furthermore, the novel data representation showed to be particularly useful to pinpoint vulnerable service parameters entangled with binary network protocols which helps security

vendors to faster develop mitigating controls.

To demonstrate applicability of proposed anomaly detection and feature extraction methods in other domains, two selected studies have been presented in the fields of malware detection and privilege misuse detection.

Chapter 7 presented a novel method for automated classification of malware behavior utilizing single-family Support Vector Machine-based classification of labeled malware based on compound string features extracted from malware behavior reports to detect novel malware and classify malware instances based on known malware families. Experiments over a corpus of 3.000 malware samples confirmed that the proposed method correctly predicted approx. 70% of labels assigned by the anti-virus solution four weeks later. Furthermore, the method also reliably identified unknown behavior, i.e. malware families not learned during training were correctly classified as "unknown". Due to the utilization of linear kernels, an analysis of discriminative features revealed interesting similarities between *Doomber| and Gobot worms, suggesting that Doomber being an extension of (Gobot*.
From the experiments, it can be concluded that compound string features extracted from malware behavior reports carry informative features to discriminate different malware families and to detect previously unknown malware instances. Furthermore, a norm-based feature analysis provided insights into characteristics shared between different malware families.

List of Figures

List of Tables

Acronyms

C2 Command & Control. 1, 3, 4

DNS Domain Name System. 1, 3

FTP File Transfer Protocol. 14

GRE Generic Routing Encapsulation. 4

HTTP Hyper-Text Transfer Protocol. 2, 3, 4, 10, 11, 14, 17, 37, 51, 59, 75, 120, 121, 123

IDS Intrusion Detection System. 2

IoT Internet of Things. 1, 4

IT Information Technology. 1

MLP Multi-layer Perceptron. 6

NAT Network Address Translation. 1

OC-SVM One-class Support Vector Machine. 6, 7, 8

RBF Radial Basis Function. 8

RPC Remote Procedure Call. 37, 120

SCADA Supervisory and Data Acquisition. 17, 37, 50, 120

SMB Server Message Block. 37

SMTP Simple Mail Transfer Protocol. 14

SQL Structured Query Language. 59, 77, 95, 121

SVDD Support Vector Data Description. 51

SVM Support Vector Machine. 6, 7, 17, 18, 36, 59, 97, 118, 121

Tbps Terabits per second. 1

TCP Transmission Control Protocol. 4

TCP/IP Transmission Control Protocol/ Internet Protocol. 27, 120

UDP User Datagram Protocol. 4

XML Extensible Markup Language. 15

XSS Cross-site Scripting. 59, 95, 121