An On-Line Learning Statistical Model to Detect Malicious Web Requests

Harald Lampesberger[1,2], Philipp Winter[1], Markus Zeilinger[1], and Eckehard Hermann[1]

[1] Upper Austria University of Applied Sciences, Department Secure Information Systems, Softwarepark 11, A-4232 Hagenberg, Austria
[2] Johannes-Kepler-University Linz, Christian-Doppler Laboratory for Client-Centric Cloud Computing, Softwarepark 21, A-4232 Hagenberg, Austria
h.lampesberger@cdcc.faw.jku.at

Abstract. Detecting malicious connection attempts and attacks against web-based applications is one of many approaches to protect the World Wide Web and its users.

In this paper, we present a generic method for detecting anomalous and potentially malicious web requests from the network's point of view without prior knowledge or training data of the web-based application. The algorithm assumes that a legitimate request is an ordered sequence of semantic entities. Malicious requests are in different order or include entities which deviate from the structure of the majority of requests. Our method learns a variable-order Markov model from legitimate sequences of semantic entities. If a sequence's probability deviates from previously seen ones, it is reported as anomalous.

Experiments were conducted on logs from a social networking web site. The results indicate that that the proposed method achieves good detection rates at acceptable false-alarm rates.

Keywords: intrusion detection, anomaly detection, on-line learning, Markov model, web security.

1 Introduction

The popularity of the Web is continuously rising and our daily lives are more and more dependent on this source of information. Accordingly, the *Hypertext Transfer Protocol* (HTTP) has evolved to one of the most employed application layer protocols in the Internet. But with increasing global dependence on the Web, attackers are even more interested in tampering with those systems.

The paper is structured as follows: The remaining introduction deals with HTTP, its security challenges and related work in this area. Section 2 explains the concept of the proposed anomaly detection method. Section 3 outlines implementation details, evaluation results are listed in Section 4 and Section 5 draws the conclusion.

M. Rajarajan et al. (Eds.): SecureComm 2011, LNICST 96, pp. 19–38, 2012.

1.1 HTTP and Web Security

The HTTP protocol [11] defines stateless and generic exchange of information. The communication is initiated by a client who requests a specific resource, identified by the Unified Resource Identifier (URI) path, from the server. The response assembles server status codes, meta information and possible entity content.

A fundamental security problem of web-based applications is that the client is out of the application's scope of control. The protocol was originally designed for static resources and stateless interaction, but today's web applications employ it for dynamic content and stateful sessions. Consequently, data sent from the client must be somehow interpreted by the server. Semantic client data can be found in the request-URI path, header fields and possible request entity content.

Request-URI Path. The path is a hierarchically structured sequence of string segments and an optional query component. The grammar is defined in [5] and traditionally, the path references a static resource, or in dynamic web applications a content generating process. A segment only allows a subset of printable characters; others must be escaped by using the URI percent-encoding.

The '?' character introduces the query component of a path and parameters are supposed to be in field-value pairs. But real-world implementations tend to break this convention because expressive path names in URLs are preferred by developers and users. This is called *URL Rewriting* and a representative example is the widely used Apache web server module *mod_rewrite* [2] which allows mapping of path segments into queries. As a result, a client can never conclude from a path whether segments are interpreted as static resources or parameters in a web-based application.

Request Headers. In a request, headers represent meta data from the client in an unordered field-value structure. For example, headers inform the server which kind of content and encoding is understood by the client. The best example for client data processed by the web application is the so-called *cookie*. Many applications use the cookie to track states in the stateless HTTP protocol.

Request Entity Content. A typical GET request can transport parameters in the URI path, but the size of the query part is restricted by server's implementation. For high-volume transmissions or forms, the POST method allows query-style or MIME-encoded data in the request entity content. Additional headers are necessary to describe type and length of this entity content.

Weaknesses. Wrong handling of client data in any function of the web application introduces a security weakness which can probably be exploited by an attacker. To name a few, attack vectors like buffer overflows, SQL or code injections, Cross-Site Scripting, Cross-Site Request Forgery or HTTP parameter pollution emerge from few common pitfalls. These classic flaws are gathered in the Common Weakness Enumeration database [22] and, in this paper, attacks

```
Legitimate Request:
  GET /fotos.php?action=view HTTP/1.1
Code Injection:
  GET /fotos.php?action=http://195.33.221.4:8081/bot.txt? HTTP/1.1
SQL Injection:
  GET /userportal.php?id=4518-999.9+union+select+0-- HTTP/1.1
Cross-Site Scripting:
  GET /fotos.php?action=search&album=%22%2F%3E%3Cscript%3Ealert%281%29
  %3B%3C%2FScript%3E HTTP/1.1
Path Traversal:
  GET /images/../../../../../../../../../../etc/passwd HTTP/1.1
```

Fig. 1. Examples for legitimate and malicious URI paths in HTTP requests

are grouped by their common weakness. Some examples are given in Figure 1 for a better understanding.

Protecting a web-based application implicitly protects its users. Drive-by downloads to create botnets are on the rise as noted by Provos et al. [25]. In addition to disturbing the service availability or stealing information from a high-volume web site, an attacker might consider planting drive-by malware to infect visitors.

1.2 Intrusion Detection

Another approach in protecting web-based applications, besides writing robust code, is the domain of payload-based intrusion detection systems (IDS) to enable prevention mechanisms or early warning. IDS techniques can be distinguished into *misuse detection* and *anomaly detection* based on the style of detection. While misuse detection relies on proper signatures of malicious behavior, anomaly detection tends to use methods such as machine learning or statistics to construct a profile of normal behavior and report deviating interactions as anomalies.

As stated by Sommer and Paxson [29], both concepts are challenged in different ways. The detection performance of misuse detection completely depends on currentness and coverage of signatures, but false-alarm rates are accordingly low. Anomaly detection is prone to costly false-alarm rates, but it is more probable by design to recognize novel attacks. To succeed in real-world scenarios, anomaly detection must consider a) the variability of input data, b) the lack of training data, c) a very high cost of errors, d) the difficulty of sound evaluation and e) descriptiveness of detection results.

Detecting malicious web requests is challenging. Encodings, especially polymorphic ones as used in attack frameworks like Metasploit [20], make it almost impossible to induce valid signatures for misuse detection. Additionally, web applications are very dynamic and constantly change over time. This *concept drift* [19] handicaps the process of learning normal behavior in anomaly detection.

1.3 Related Work

Anomaly detection in network data is not new. Over the past years, different strategies for extracting representative features from network payload were presented. The payload-based anomaly detector (*PAYL*) by Wang and Stolfo [34] uses byte frequencies for payload profiling. *Anagram* [33] is an advancement of PAYL using n-grams instead of single byte frequencies. Perdisci et al. [24] further pursue this approach and introduce McPAD, a method based on 2_ν-grams. PAYL, Anagram and McPAD are rather generic concepts to analyze application layer network traffic, but their evaluation focuses on HTTP. Also, the three methods rely on training data sets.

Kruegel and Vigna [16] introduce the first detection system focused on web applications. It uses a linear combination of six different anomaly detection measures like attribute character distributions, structural information or attribute lengths. This concept establishes the foundation for follow-up research: grouping similar anomalies [26], addressing concept drift [19] and dealing with scarce training data [27].

Ingham et al. [14] define an approach where finite automaton representations are learned from HTTP tokens. Another method customized to protocol syntax using an attributed token kernel in One-Class Support Vector Machines is shown by Duessel et al. [9]. *Spectogram* is a model of multiple Markov chains proposed by Song et al. [30]. Ma et al. [18] define a model based on compression for web anomaly detection which tolerates concept drift to a certain degree. The HTTP reverse proxy *TokDoc*, presented by Krueger et al. [17], uses an ensemble of anomaly detection methods to detect, and automatically repair, malicious web requests.

All previously listed algorithms achieve good evaluation results, but they depend on training data. Especially for a fast-paced large-scale web application it is hardly possible to create an up-to-date and representative training data set. Görnitz et al. [12] realize this problem and present an active learning strategy based on methods such as PAYL, Anagram and McPAD. Their solution actively queries for labels to reduce the need for training, but context drift is not addressed.

1.4 Scope of This Work

As outlined in 1.1, a client or network device cannot conclude which elements of a web request will be processed in weakness-prone functions of the web application. Furthermore, URL Rewriting is not mentioned in previous work, but it is actively used in practical scenarios. So, RFC-compliant queries in URI paths cannot be assumed. The only possible assumption is that during normal operation of a web site, the application is probably receiving more legitimate web requests than malicious ones.

This paper explores the question of whether potentially malicious web requests can be detected from the network's point of view without prior knowledge at decent performance levels. An exemplary implementation scenario is a network-based IDS system for providers to monitor high-volume web sites and provide early warning mechanisms. Considering Sommer and Paxson's conclusions [29], the following requirements were defined:

– No explicit training data is necessary,
– The model considers concept drift of the web application,
– The model accepts URL Rewriting,
– False-alarm rates are minimized,
– Details on an alert's cause are available and
– Throughput performance is kept in mind.

2 Methodology

Our approach is formed by two assumptions:

– A legitimate web request is a series of semantic entities in specific order and
– Normal requests are more probable than malicious ones.

Within a web request, the data, especially the URI path, is in some kind of order as the result of design principles. For example, if the request is processed as stream and the first bytes indicate request method POST, then entity content is to be expected. It might come natural to say that a web request is *Markovian*. A conjecture of our detection method is that malicious requests have unexpected order of data or include entities which differ from the common structure. Consequently, Markov modeling seems to be a suitable approach for prediction.

It is important to mention that web-based applications actually use random strings in requests, for example session identifiers, random file names of image thumbnails, random transaction codes and so on. If transition probabilities purely rely on single byte frequencies, a single Markov model will get falsified by random strings. But Markov chains and Hidden Markov Models have successfully been employed for modeling web requests [19, 16, 26, 30, 18, 17, 27]. These concepts use multiple models to cope with high-entropy content.

Our method was inspired by Begleiter et al. [4] and their work on sequence prediction using compression models. The core idea is to deduce a variable-order Markov model (VMM) from legitimate web requests and use this model to classify novel web requests based on their probability. To increase robustness and handle high-entropy content, the grammar of HTTP is exploited to transform a web request into a sequence of abstract symbols beforehand. A novel web request is classified whether it is normal or not by comparing its probability to the distribution of ones. Therefore, the algorithm maintains a sliding window over recent sequence probabilities.

Based on Vovk et al. [31, pp. 3–7], we consider learning in our scenario as *transductive on-line learning*: Instead of inducing a general rule from training data, samples are presented one by one to the model, it predicts the sample's label and adds it to a bag of training examples. In our case, a so-called sample equals a web request and the VMM represents the bag of training examples. Each new prediction relies on previously seen samples, no induction is needed and the quality of predictions should improve over time. Finally, the model predicts whether a sample is normal or anomalous.

On-line learning requires somehow feedback of the truth. Following Vovk's definitions [31, p. 107], our scenario has two so-called *lazy teachers* who occasionally reveal the true label of a sample. The first lazy teaching mechanism is a constrained randomness assumption: the majority group of similar samples is probably normal. The second lazy and slow teacher is the human expert who works with and maintains the system. The expert intervenes after possible delay if false positive or false negative detections were realized. To sum up, our proposed algorithm processes a web request in four steps:

1. The web request is converted into a sequence of symbols,
2. A VMM estimates a probability of the sequence,
3. The sequence probability is assigned to a confidence interval in the sliding window probability distribution and
4. Depending on the confidence interval, the sequence is learned, ignored or reported.

2.1 A Request is a Sequence of Symbols

RFC 2616 [11] defines the grammar of the US-ASCII-oriented HTTP. The fundamental grammar entity is one byte. A class of characters called *separators* has special meaning in the protocol. So, in context of this work, a symbol $\sigma \in {\mathbb{N}_0}^{16}$ is the statistical representation of bytes between two separators. These 16 occurrence counters of a symbol are a computationally optimized heuristic to model the appearance of a variable-length string token in fixed-length memory while simultaneously handling high-entropy content. The counter definitions are based on HTTP character classes defined in the RFC and some additional counters capture structural characteristics of the content:

$$\sigma \Rightarrow \begin{cases} \sigma[0] & \text{amount of printable ASCII characters,} \\ \sigma[1-4] & \text{lexical letter index} \in \{a..z, A..Z\} \mod 4, \\ \sigma[5-6] & \text{digit index} \in \{0..9\} \mod 2, \\ \sigma[7] & \text{uppercase letters} \in \{A..Z\}, \\ \sigma[8] & \text{lowercase letters} \in \{a..z\}, \\ \sigma[9] & \text{US-ASCII control characters,} \\ \sigma[10] & \text{protocol-specific bytes} \in \{\texttt{CR LF SPACE TAB}\}, \\ \sigma[11] & \text{path-specific characters} \in \{\texttt{./}\}, \\ \sigma[12] & \text{protocol separators} \in \{\texttt{?\&;()<>@,:[]\{\}=\textbackslash}\}, \\ \sigma[13] & \text{single and double quotes,} \\ \sigma[14] & \text{percent character,} \\ \sigma[15] & \text{non-US-ASCII character.} \end{cases} \tag{1}$$

A web request is transformed into an n-tuple or sequence of symbols $q_1^n = (\sigma_1, \sigma_2, \ldots, \sigma_n)$. For precise tokenization of URI paths, the class of separators is split up in pre- and post-separators:

$$\begin{aligned} \text{pre-separators} &= \{\texttt{SPACE TAB}\}, \\ \text{post-separators} &= \{\texttt{/?\&;()<>@,:"[]\{\}=\textbackslash}\}. \end{aligned} \tag{2}$$

In a data stream, the observation of a pre-separator triggers the allocation of a new symbol in the sequence before the observed byte increments the occurrence counter in the symbol. Given the previous definitions, Figure 2 shows part of an exemplary HTTP GET request and how it is transformed into a sequence of symbols.

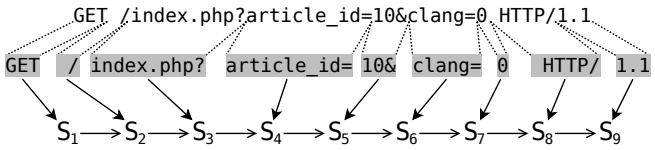


Fig. 2. Transformation of a web request data into a sequence of symbols

The dynamic alphabet $\mathcal{A}$ consists of all the symbols the anomaly detection model is currently aware of. The initial alphabet is empty and as a result of learning, symbols are added or removed over time. For prediction, the symbols of the web request are mapped to similar symbols in $\mathcal{A}$ if possible. This mapping function Φ requires a metric to compare symbols. A similarity measure between two sets is the *Tanimoto coefficient* τ [13, p. 398] which estimates the intersection of two symbols:

$$\tau(\sigma_1, \sigma_2) = \frac{\sigma_1 \cdot \sigma_2}{\|\sigma_1\|^2 + \|\sigma_2\|^2 - \sigma_1 \cdot \sigma_2} \,. \tag{3}$$

Let $T_{\mathcal{A}}$ be the similarity threshold for alphabet $\mathcal{A}$ and also an anomaly detection model parameter. Two symbols σ_1 and σ_2 are considered identical if $\tau(\sigma_1, \sigma_2) > T_{\mathcal{A}}$. So, the mapping function Φ is defined as:

$$\Phi(\sigma, \mathcal{A}, T_{\mathcal{A}}) = \begin{cases} \arg\max\limits_{\nu \in \mathcal{A}} \tau(\sigma, \nu) & \text{if } \exists \nu \in \mathcal{A} : \ \tau(\sigma, \nu) > T_{\mathcal{A}}, \\ \sigma & \text{otherwise.} \end{cases} \tag{4}$$

2.2 Prediction by Partial Matching

The idea of *Prediction by Partial Matching* (PPM) is to predict the next symbol $\sigma \in \mathcal{A}$ in a stream based on the previously seen symbols, the so-called context $s \in \mathcal{A}^n$ of order n. Probability estimates are based on symbol counts in the data.

Cleary and Witten [7] present PPM as a concept of statistical modeling for lossless compression. PPM belongs to the group of variable-order Markov models which are able to capture both large and small order Markov dependencies in observed data, as stated by Begleiter et al. [4]. To handle the zero-frequency problem when novel symbols are encountered, PPM provides the *escape* and *exclusion* mechanisms. In this work, exclusion is ignored due to the computational overhead and escape follows 'Method C' proposed by Moffat [23].

PPM requires an upper Markov order bound D for VMM construction. A data structure to model PPM is a *trie* of depth $D + 1$. A trie node references

a symbol from alphabet $\mathcal{A}$ and maintains a frequency counter. Each path from root to node represents a subsequence in the already processed stream and the node's count indicates, how often this subsequence appeared. Figure 3 shows an exemplary trie for Markov order $D = 2$ constructed from a single sequence.

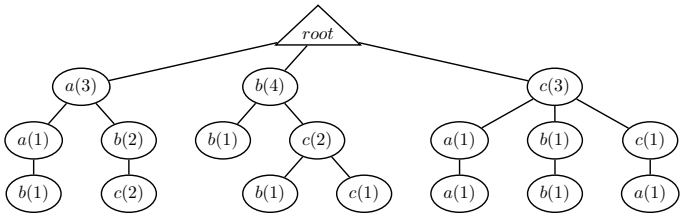


Fig. 3. PPM trie (order $D = 2$) for simplified sequence $q_1^{10} = abccaabcbb$

Let k be the length of the current context and $k \leq D$. Estimating the probability $\hat{P}$ of symbol σ considering its context s follows a recursive relation, where s' is a one-symbol-shorter context and $k < 0$ ends the recursion:

$$\hat{P}(\sigma|s) = \begin{cases} \hat{P}_k(\sigma|s) & \text{if } s\sigma \text{ exists in the VMM,} \\ \hat{P}_k(escape|s) \cdot \hat{P}(\sigma|s') & \text{otherwise.} \end{cases} \tag{5}$$

Let $\mathcal{A}_s$ be the specific alphabet of context s and $N(s\sigma)$ be a count value of the node in context s referencing symbol σ. Then the probability estimates based on 'Method C' [23] are:

$$\hat{P}_k(\sigma|s) = \frac{N(s\sigma)}{|\mathcal{A}_s| + \sum\limits_{\sigma' \in \mathcal{A}_s} N(s\sigma')} \quad \text{if } \sigma \in \mathcal{A}_s \ , \tag{6}$$

$$\hat{P}_k(escape|s) = \frac{|\mathcal{A}_s|}{|\mathcal{A}_s| + \sum\limits_{\sigma' \in \mathcal{A}_s} N(s\sigma')} \quad \text{otherwise.} \tag{7}$$

The escape probability depends on the entropy within a specific context and the alphabet size is not assumed finite. The following examples based on Figure 3 are for a better understanding of the estimation process: The probability that a sequence starts with b is $\hat{P}(b) = \frac{4}{3+10} = 0.308$. The chance that c occurs after 'ab' is $\hat{P}(c|ab) = \frac{2}{1+2} = 0.667$. But a after 'bc' has not been seen before, so $\hat{P}(a|bc) = \hat{P}(escape|bc) \cdot \hat{P}(a|c) = \frac{2}{2+2} \cdot \frac{1}{3+3} = 0.083$.

Finally, the average probability of a sequence q_1^n is the arithmetic mean of all its symbol probabilities:

$$\hat{P}(q_1^n) = \frac{1}{n} \sum_{i=1}^{n} \hat{P}(q_i|q_{i-D}^{i-1}) \ . \tag{8}$$

In context of web-based applications, a perfect VMM that learned all possible web requests delivers approximately high mean probability scores for legitimate sequences. A malicious web request will likely contain symbols that are unknown to the VMM's alphabet or symbol arrangements in an unexpected order. This results in a low mean probability score of the sequence. The distribution of probabilities depends on the web application and its dynamics. Accordingly, a static threshold for classification of outliers is insufficient.

2.3 Detecting Outliers

The proposed outlier detection method assumes that mean sequence probabilities of all legitimate web requests are somehow similar distributed in a perfect VMM. Different quantiles of the estimated distribution represent confidence intervals. Outliers are found in intervals distant to the mean.

Algorithm 1. Sliding window mean and sample variance estimator in $O(1)$

Require: $w_{size} > 0$
  empty queue $W \leftarrow [\,]$
  sum-of-squared residuals $M_2 \leftarrow 0$
  fill count $n \leftarrow 0$
  $\bar{P} \leftarrow s^2 \leftarrow 0$
  while $P_{new} \leftarrow Input$ **do**
    if $n < w_{size}$ **then**
      $n \leftarrow n + 1$
    else
      $P_{old} \leftarrow Dequeue(W)$
      $\delta \leftarrow P_{old} - \bar{P}$
      $\bar{P} \leftarrow \bar{P} - \delta/(n-1)$
      $M_2 \leftarrow M_2 - \delta * (P_{old} - \bar{P})$
    end if
    $\delta \leftarrow P_{new} - \bar{P}$
    $\bar{P} \leftarrow \bar{P} + (\delta/n)$
    $M_2 \leftarrow M_2 + \delta * (P_{new} - \bar{P})$
    $Enqueue(W, P_{new})$
    if $n > 1$ **then**
      $s^2 \leftarrow M_2/(n-1)$
    end if
    print $\bar{P}, s^2$ {mean and sample variance of previous w_{size} entities}
  end while

The bounded probability space $[0, 1]$ is supported by the Beta distribution $Beta(\alpha, \beta)$. The parameters for this distribution are estimated from the mean $\bar{P}$ and sample variance s^2 of recent sequence probabilities by the method-of-moments [10, p. 40]:

$$\hat{\alpha} = \bar{P}\left(\frac{\bar{P}\,(1-\bar{P})}{s^2} - 1\right), \qquad \hat{\beta} = (1-\bar{P})\left(\frac{\bar{P}\,(1-\bar{P})}{s^2} - 1\right). \tag{9}$$

Due to numerical and complexity boundaries, it is challenging to calculate the mean and sample variance in a streaming scenario, where each new sequence causes an update of the values. Maintaining a sliding window over the recent w_{size} sequence probabilities reduces the computational complexity. Also, a sliding window forgets values over time and allows better adaption to concept drift of the underlying application. The size of the sliding window affects how strong the mean and sample variance are affected by outliers in the data.

For computational efficiency, the algorithm for one-pass mean and sample variance estimation proposed by Welford [35], and recommended by Knuth [15, p. 216], has been modified for sliding windows. Algorithm 1 updates the sliding window mean $\bar{P}$ and sample variance s^2 in constant time. All probability values stay in a FIFO queue for w_{size} updates and before discarding them, their moments are withdrawn from the mean and sample variance to attain the sliding window.

The confidence c_q of a web request's mean sequence probability $\hat{P}(q)$ is estimated by the Beta distribution's cumulative distribution function:

$$c_q = I_{\hat{P}(q)}(\hat{\alpha}, \hat{\beta}). \tag{10}$$

We define three confidence thresholds as model parameters: base confidence T_{base}, warn confidence T_{warn} and alert confidence T_{alert}. As a result, four confidence intervals are formed in the distribution and Figure 4 outlines them. A web request is classified according to its confidence c_q:

$$classify(c_q) = \begin{cases} \textit{Normal} \text{ (learning)} & \text{if } c_q > 1 - T_{base}, \\ \textit{Normal} \text{ (ignore)} & \text{if } 1 - T_{base} > c_q > 1 - T_{warn}, \\ \textit{Anomalous} \text{ (warn)} & \text{if } 1 - T_{warn} > c_q > 1 - T_{alert}, \\ \textit{Anomalous} \text{ (alert)} & \text{otherwise.} \end{cases} \tag{11}$$

To sum up outlier detection, a Beta probability distribution over previous VMM prediction results is estimated. Depending on a web request's confidence, the grade of abnormality is known, it is assigned to one of four confidence intervals and further learning or reporting actions are taken.

2.4 On-line Learning Strategy

The VMM requires learning of legitimate sequences to reduce VMM escapes and to increase prediction precision over time. Better predictions result in higher mean and lower sample variance, the distribution and its confidence intervals get more and more distinct and anomaly detection performance improves.

Learning. The first lazy teacher in the on-line learning scenario is a constrained randomness assumption: most of the web requests are probably normal. Consequently, sequences in the learning interval are automatically fed back, the VMM trie grows and new symbols are added to alphabet $\mathcal{A}$.

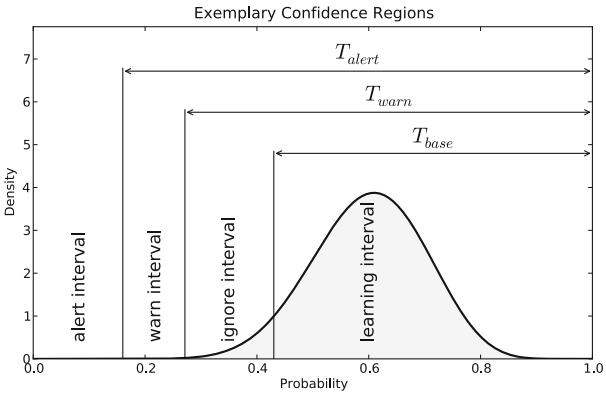


Fig. 4. Exemplary Beta probability-density function graph where the four confidence intervals (*alert*, *warn*, *ignore* and *learning interval*) are marked

The second lazy and slow teacher is a human expert who eventually recognizes a false positive or false negative with possible delay. In the case of false positive, the sequence and novel symbols are added into the VMM trie and alphabet. The according node counters are incremented until the sequence resides in the learning interval. If a false negative detection is corrected, the trie nodes related to the sequence are decremented or removed from the trie. Unreferenced alphabet symbols are deleted.

Especially during the first hundred web requests, a malicious attempt may unintentionally be learned. Also because of concept drift, the web-based application might change, new resources appear or old resources fade to exist. The web application matures and the detection model must *forget* outdated information over time too.

Pruning. Due to concept drift and numerical limits in computers, the VMM trie and its counters cannot grow indefinitely. The model parameter T_{prune} is a threshold for the most frequent node counter in the trie. If the most frequent node exceeds T_{prune}, pruning is performed. All node counters in the trie are integer divided by two, zero nodes or branches are removed and unreferenced symbols are deleted from the alphabet.

So, VMM escape probabilities increase again, the model is able to adapt to a certain degree of concept drift and malicious sequences learned by mistake will be dropped over time.

To sum up all introduced model parameters, the proposed anomaly detection model M is parameterized by:

$$M\langle T_{\mathcal{A}}, D, w_{size}, T_{base}, T_{warn}, T_{alert}, T_{prune}\rangle \ .$$

3 Implementation

The proposed methodology is implemented in two independent prototypes with the same algorithmic background: an off-line log file analyzer for performance evaluation and a passive network analysis tool. For performance reasons, all implementations are written in C and the efficient trie data structure follows the recommendations from Salomon [28, pp. 150–155].

3.1 Network Operation

The network prototype is built upon Libnids [36], a library for payload inspection of TCP sessions in live network traffic or recordings. Due to full decoding of TCP sessions, the library is resistant to fragmentation attacks. Furthermore, it allows intervening in established TCP sessions by sending forged reset segments to both communication partners. Reset segments are an unreliable third-party method for killing connections because of possible network delays, but it still gives this prototype some intrusion prevention abilities.

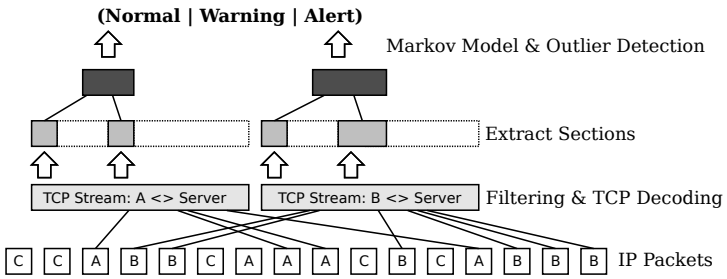


Fig. 5. Analysis concept for network data processing

Figure 5 outlines the concept of network data processing. Libnids decodes TCP sessions for a configured subset of destination hosts, other sessions are ignored. Packet payload is handed to the *Protocol State Machine* (PSM). The PSM is a TCP session-specific deterministic finite automaton, where state transitions are triggered by payload byte tokens. A transition also performs user-specified actions. This includes starting and finalizing of anomaly detection, reporting, killing connections or canceling further analysis of the session.

The PSM states, transition tokens and actions are defined in XML by the user and Aho-Corasick pattern matching [1] enables the search for these tokens in the payload stream. So, the computationally intensive anomaly detection can be limited to weakness-prone sections in the protocol, for example the HTTP request and response headers.

A TCP session is reported if it is anomalous. The raw analyzed data and prediction results are kept in a ring-buffer for a certain amount of time. In case of a detection error, an expert can see which symbols in the payload stream are responsible for the anomaly.

At last, all anomaly detection model parameters are changeable during operation. The network prototype features an XML-RPC interface for parameter modifications or teaching of false positive or false negative detections.

4 Experiments

For evaluation of detection performance, we assume a binary classification case where legitimate requests represent class *Normal* and warnings or alerts are considered as class *Attack*. A labeled data set is required to construct a confusion matrix as shown in Table 1. The values in the matrix are mandatory for estimating performance metrics.

Table 1. Confusion matrix for the binary classification case

		Actual	
		Attack	**Normal**
Predicted	**Attack**	True Positive (TP)	False Positive (FP)
	Normal	False Negative (FN)	True Negative (TN)

The Receiver Operator Characteristic (ROC) curve and its area under the curve (AUC) are commonly used metrics to describe detection performance of a classification algorithm. But in the intrusion detection area, normal and malicious examples are not equally distributed. So, false positives cause a much higher cost and impact in the IDS area, as already shown by Axelsson [3]. We assume that ROC is not an optimal choice in this case.

Performance evaluation in this paper uses the metrics *Precision* and *Recall* as recommended by Davis and Goadrich [8] for skewed data sets. For intrusion detection, Recall is equivalent to detection rate and Precision indicates how reliable the detections are. The Precision-Recall (PR) curve and its area under the curve (PR-AUC) give better information on the algorithm's performance in a scenario, where examples are not equally distributed. Also, Precision and the false positive rate (FPR) are interdependent. Maximizing Precision implicitly minimizes the FPR.

$$Precision = \frac{TP}{TP + FP}, \quad Recall = \frac{TP}{TP + FN}, \quad FPR = \frac{FP}{FP + TN}. \tag{12}$$

In PR space, a perfect algorithm has maximum Precision for the complete Recall range, the curve is in the upper-right corner and PR-AUC = 1. The PR-AUC represents the capability of an algorithm to correctly separate the two classes in the binary classification case.

4.1 Evaluation Data

Realistic data is mandatory for sound evaluation. As logs contain a part of the web request, the presented evaluation results are based on anonymized web site

log files. A data set is neither partitioned for training nor ordered, the analysis starts with an empty model and the first sample, and ends with the last sample in the data set. So, these experiments are kept as realistic as possible.

Manifesting attacks are planted randomly in the data sets. Table 2 shows a pool of 57 unique attack vectors and their CWE classes. Custom attacks are adapted to the web application in use; others are referenced either by their Common Vulnerability and Exposures (CVE) [21] identifier or worm name.

Table 2. Unique CWE weaknesses for a total of 57 attack vectors

CWE	Name	Num	CVE or Other References
20	Input Validation	16	worm:Nimda worm:CodeRed
22	Path Traversal	3	custom:2 2010-2334
78	OS Command Injections	5	custom:3 2005-0116 2005-2847
79	Cross-Site Scripting	8	custom:5 2010-0804 2010-2356 2010-4366
89	SQL Injection	9	custom:3 2005-1810 2008-0397 2008-1982 2009-0968 2010-3601 2011-0519
94	Code Injection	5	custom:3 2005-0511 2007-1599
119	Buffer Errors	11	1999-0874 2001-0241 2001-0500 2003-0109 2003-1192 2004-1374 2004-1561 2004-1134 2006-1148 2006-5216 2007-0774

CMS Data Set. The data is from a PHP-based content management system named Redaxo and samples were collected within several months. The original data contains 108 malicious attempts, basically automated scans and code injections. The final set consists of 3,279 log lines where additional 29 attacks are added.

CACTI Data Set. Samples are from the web front-end of a Cacti monitoring solution deployed in a hosting environment and were collected within approximately one month. There is one code injection attempt in the original data and it is free of scanning events. The final set with planted attacks has 25,057 request samples where 126 requests are malicious.

SOCIAL Data Set. The log data is from a social networking site which is a hybrid solution of different web applications. From the analyst's point of view, the data is a worst-case scenario because there is a) concept drift, b) user data like events or names in the URI path, c) URL Rewriting, d) lots of random data like names of image thumbnails and e) an advertising system that transmits the encoded referee URL within the URI path.

The original set has 12,515,970 log lines and contains 1,922 attacks where 1,392 are scanning attempts. Also, 115 suspicious requests are the result of a JavaScript fault in the application and marked as CWE-0 in this paper. This data was collected in a timespan of about two weeks. The final data set for evaluation has 12,528,513 samples where a total of 14,465 are considered anomalous.

Table 3. Distribution of weaknesses in the data sets

CWE	0	20	22	78	79	89	94	119	200	total	fraction
CMS	0	7	1	2	6	6	58	5	52	137	4.178%
CACTI	0	26	7	18	30	15	8	22	0	126	0.503%
SOCIAL	115	3464	611	1137	1794	2076	1416	2460	1392	14465	0.115%

4.2 Results

To keep experiments as realistic as possible, we assume that a virtual expert gives feedback to the algorithm occasionally. This expert randomly recognizes 66.6% of false positives and 10% of false negatives and triggers a learning function. All experiments were performed on one core of a consumer-grade Intel i5-760 CPU.

The advantage of the chosen scenario is that it is oriented on practical deployment. Due to the constant on-line learning and varying detection performance, results cannot be directly compared to solutions that are pre-trained on existing training data.

CMS Results. The CMS data set is a toy example to visualize outlier detection and learning. Figure 6 shows the time-series of evaluated samples. Within the first 500 requests, the confidence intervals stabilize. As visible at about sample 1,000, only few values are in the sliding window, the distribution is not yet robust against outliers.

A model with parameters $\langle 0.7, 2, 10000, 0.99, 0.999, 0.9999, 50000 \rangle$ minimizes false positives to one and achieves Recall = 97.08% and Precision = 99.25%. The final model has 119 trie nodes, alphabet size $|\mathcal{A}| = 15$ and reaches throughput of 91,083 logs/second due to the simpleness of the underlying web application.

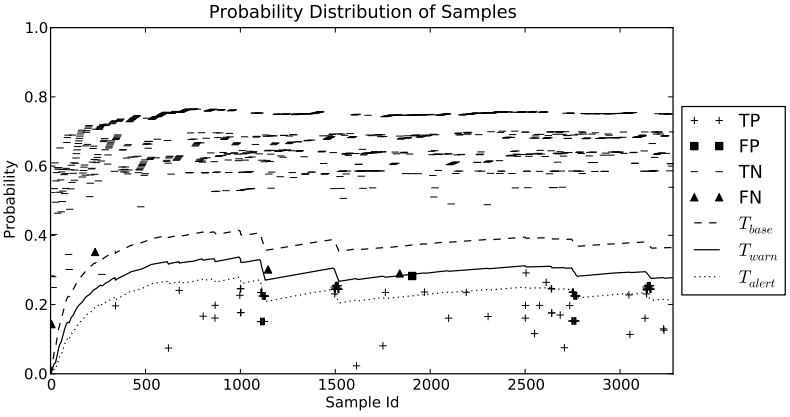
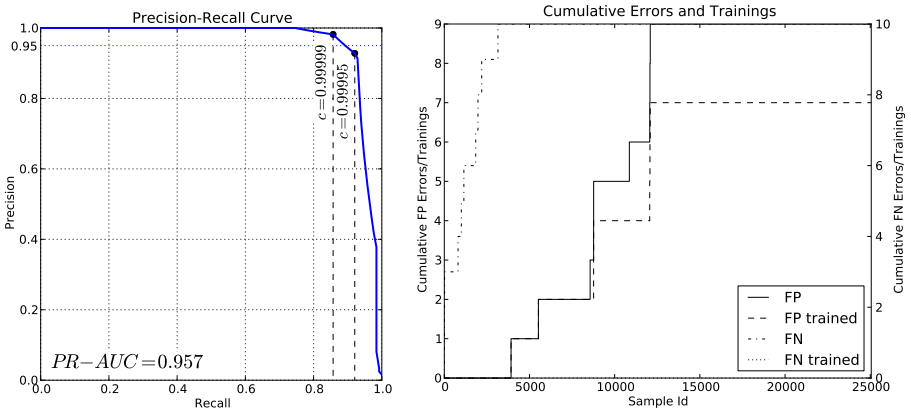


Fig. 6. Time-series of sequence probabilities and evolution of confidence intervals in the CMS data set



(a) PR curve. (b) Cumulative error and training curves.

Fig. 7. Performance metrics for the CACTI data set

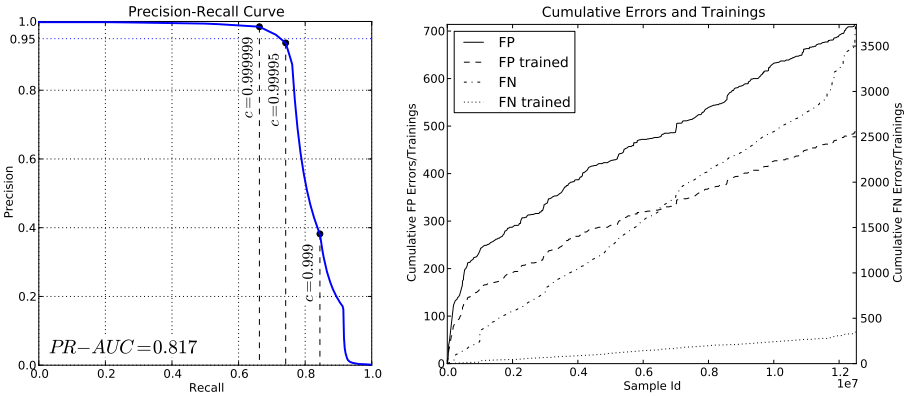
CACTI Results. This data set is more realistic and performance curves are displayed in Figure 7. Here, Recall = 92.06% and Precision = 92.8% are achieved by parameters $\langle 0.82, 2, 10000, 0.995, 0.99995, 0.99995, 10000\rangle$, only nine false positives take place in the simulated timespan.

Figure 7(a) outlines the PR curve and two different confidence thresholds are marked. It is visible that an increased threshold also increases the precision at the expense of detection rate. The cumulative curves in Figure 7(b) show that false negatives only occur in the initial phase and after about sample 13,000 the growth of false positives stagnates. This stagnation indicates that the statistical model adepts to the data. After the last sample processed, the model has 225 trie nodes, alphabet size $|\mathcal{A}| = 20$ and still achieves throughput of 65,083 logs/second.

SOCIAL Results. The last data set represents the worst case experiment and resulting performance curves are shown in Figure 8. A model with parameters $\langle 0.8, 4, 20000, 0.995, 0.99995, 0.99995, 5000000\rangle$ achieves the best performance with Recall = 74.15% and Precision = 93.76%. A total of 714 false positives yield FPR = $5.71 \cdot 10^{-5}$. The two least-recalled classes of weaknesses are scanning attempts and the already mentioned JavaScript fault.

Figure 8(b) outlines, that most false positive detections take place in the initial phase and growth decreases over time. Due to the complexity of this web application, the final model has 19,650 trie nodes, alphabet size $|\mathcal{A}| = 100$ and permits throughput of 29,200 logs/second.

To sum up, the results of all three data sets are promising considering the on-line scenario and evaluation data. Also, the search for optimal performance has shown that initial parameters $\langle 0.7, 2, 10000, 0.99, 0.9999, 0.9999, 500000\rangle$ are a good start. For each data set there are several parameter combinations with comparable performance results and the presented ones in this paper maximize throughput. The parameter $T_{\mathcal{A}}$ has direct impact on the size of the alphabet and accordingly, the throughput performance.



(a) PR curve. (b) Cumulative error and training curves.

Fig. 8. Performance metrics for the SOCIAL data set

4.3 Evasion Strategies

The proposed concept relies on statistical features collected over time to detect deviating web requests. Due to the nature of the problem domain, an attacker with detailed knowledge about the algorithm might be able to evade detection under certain conditions. Three potential evasion strategies, which apply to the proposed algorithm, have been studied in theory.

Initial Phase Attack. During the initial phase of deployment, the algorithm might unintentionally learn an attack. If this attack keeps undetected and similar attacks occur regularly, the algorithm will assume them as normal too. In the best case, the attack is a single incident and the pruning mechanism will clean the VMM and symbol alphabet over time. In the worst case, the web application constantly receives a high amount of similar malicious requests. This scenario needs lower T_{base} and T_{prune} thresholds to limit the feedback of sequences into the VMM. As a side-effect, these parameters will produce more false positive detections and require more human expert feedback, especially during the initial phase.

Mimicry Attack. A skilled attacker might be able to craft malicious data which undergoes detection [32]. The presented algorithm has shown to be resistant against classic polymorphic attacks, but a potential weakness is the decision-making based on the arithmetic mean sequence probability. In a malicious request, some symbols have a very low probability, and so, the mean sequence probability is lowered towards zero. But if the attacker is able to extend the malicious request with additional highly probable symbols, the impact of low-probable symbols on the mean decreases and the attack might not be recognized. A possible countermeasure is to increase the algorithm's sensitivity by increasing order D and $T_{\mathcal{A}}$ while reducing T_{warn} and T_{alert}.

Frog-Boiling Attack. This category of poisoning attack [6] affects the presented detection mechanism. It aims to falsify the statistical detection model by continuously sending borderline legitimate requests. At some point, the detection model will be too inaccurate to detect real attacks. A possible countermeasure, in addition to increasing the algorithm's sensitivity, is to include the server's response into the analysis. For example, tampering with the URI path will likely produce invalid requests, and accordingly, bad response codes. The downside of using response codes for decision-making is the limitation of prevention capabilities, because the malicious data has already been sent. This concept has been implemented in the network prototype, but more testing is still required.

5 Conclusion and Future Work

We propose an on-line learning approach to detect malicious web requests. The main contribution of this paper is a concept that addresses both concept drift of web applications and the problem of representative training data. Also by design, the algorithm copes with URL Rewriting which is popular in realistic web deployments. In experiments with realistic log data the implemented log analyzer prototype shows decent detection and throughput performance.

To sum up, our presented method transforms the HTTP request into a sequence of symbols, where one symbol is the statistical representation of bytes between HTTP separator characters. A variable-order Markov model assigns a probability of occurrence to the sequence. An estimated Beta distribution over recent sequence probabilities is used to detect deviating sequences. In case of an detected anomaly, an expert can trace the responsible section in the web request according to the individual symbol probabilities. Feedback of highly probable sequences into the model achieves lazy teaching in context of on-line learning, also, the human expert can intervene in case of erroneous detections.

For future research, testing the network prototype implementation on real-world network data is necessary. This includes comparison to other existing methods and long-term testing. Also, binary classification is insufficient for practical scenarios because the abnormality of an alert does not reflect its potential impact. For example, scanning attempts are not as harmful as successful code injections. Clustering of similar alerts is a reasonable approach here. Furthermore, throughput performance can still be optimized if parallelization or GPU-offloading is considered.

References

1. Aho, A.V., Corasick, M.J.: Efficient string matching: an aid to bibliographic search. Commun. ACM 18(6), 333–340 (1975)
2. Apache 2.0 Documentation: Apache Module mod_rewrite (2011), `http://httpd.apache.org/docs/2.0/mod/mod_rewrite.html` (Online; accessed April 28, 2011)

3. Axelsson, S.: The base-rate fallacy and its implications for the difficulty of intrusion detection. In: CCS 1999: Proceedings of the 6th ACM Conference on Computer and Communications Security, pp. 1–7. ACM, New York (1999)
4. Begleiter, R., El-Yaniv, R., Yona, G.: On prediction using variable order markov models. J. Artif. Int. Res. 22(1), 385–421 (2004)
5. Berners-Lee, T., Fielding, R., Masinter, L.: Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (Standard) (January 2005), http://www.ietf.org/rfc/rfc3986.txt
6. Chan-Tin, E., Feldman, D., Hopper, N., Kim, Y.: The Frog-Boiling Attack: Limitations of Anomaly Detection for Secure Network Coordinate Systems. In: Chen, Y., Dimitriou, T.D., Zhou, J. (eds.) SecureComm 2009. LNICST, vol. 19, pp. 448–458. Springer, Heidelberg (2009)
7. Cleary, J.G., Witten, I.H.: Data compression using adaptive coding and partial string matching. IEEE Transactions on Communications 32, 396–402 (1984)
8. Davis, J., Goadrich, M.: The relationship between precision-recall and roc curves. In: ICML 2006, pp. 233–240. ACM, New York (2006)
9. Düssel, P., Gehl, C., Laskov, P., Rieck, K.: Incorporation of Application Layer Protocol Syntax into Anomaly Detection. In: Sekar, R., Pujari, A.K. (eds.) ICISS 2008. LNCS, vol. 5352, pp. 188–202. Springer, Heidelberg (2008)
10. Evans, M., Hastings, N., Peacock, B.: Statistical Distributions, 3rd edn. Wiley-Interscience (2000)
11. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard) (June 1999), http://www.ietf.org/rfc/rfc2616.txt, updated by RFCs 2817, 5785
12. Görnitz, N., Kloft, M., Rieck, K., Brefeld, U.: Active learning for network intrusion detection. In: Proceedings of the 2nd ACM Workshop on Security and Artificial Intelligence, AISec 2009, pp. 47–54. ACM, New York (2009)
13. Han, J., Kamber, M.: Data Mining: Concepts and Techniques, 2nd edn. Morgan Kaufmann Publishers Inc., San Francisco (2006)
14. Ingham, K.L., Somayaji, A., Burge, J., Forrest, S.: Learning dfa representations of http for protecting web applications. Comput. Netw. 51, 1239–1255 (2007)
15. Knuth, D.E.: The Art of Computer Programming. Seminumerical Algorithms, 2nd edn., vol. II. Addison-Wesley (1981)
16. Kruegel, C., Vigna, G.: Anomaly detection of web-based attacks. In: CCS 2003: Proceedings of the 10th ACM Conference on Computer and Communications Security, pp. 251–261. ACM, New York (2003)
17. Krueger, T., Gehl, C., Rieck, K., Laskov, P.: Tokdoc: a self-healing web application firewall. In: SAC 2010: Proceedings of the 2010 ACM Symposium on Applied Computing, pp. 1846–1853. ACM, New York (2010)
18. Ma, J., Liu, X., Wang, Q., Dai, G.: Compression-based web anomaly detection model. In: 2010 IEEE 29th International Performance Computing and Communications Conference (IPCCC) (December 2010)
19. Maggi, F., Robertson, W., Kruegel, C., Vigna, G.: Protecting a Moving Target: Addressing Web Application Concept Drift. In: Kirda, E., Jha, S., Balzarotti, D. (eds.) RAID 2009. LNCS, vol. 5758, pp. 21–40. Springer, Heidelberg (2009)
20. Metasploit: The Metasploit Project (2011), http://www.metasploit.com/ (Online; accessed April 30, 2011)
21. MITRE Corporation: Common Vulnerabilites and Exposures (2011), http://cve.mitre.org/ (Online; accessed May 12, 2011)
22. MITRE Corporation: Common Weakness Enumeration (2011), http://cwe.mitre.org/ (Online; accessed April 28, 2011)

23. Moffat, A.: Implementing the ppm data compression scheme. IEEE Transactions on Communications 38(11), 1917–1921 (1990)
24. Perdisci, R., Ariu, D., Fogla, P., Giacinto, G., Lee, W.: Mcpad: A multiple classifier system for accurate payload-based anomaly detection. Computer Networks 53(6), 864–881 (2009); traffic Classification and Its Applications to Modern Networks
25. Provos, N., McNamee, D., Mavrommatis, P., Wang, K., Modadugu, N.: The ghost in the browser analysis of web-based malware. In: Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets. USENIX Association, Berkeley (2007)
26. Robertson, W., Vigna, G., Kruegel, C., Kemmerer, R.: Using generalization and characterization techniques in the anomaly-based detection of web attacks. In: Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, CA (February 2006)
27. Robertson, W., Maggi, F., Kruegel, C., Vigna, G.: Effective anomaly detection with scarce training data. In: Proceedings of the Network and Distributed System Security Symposium (NDSS), San Diego, CA (February 2010)
28. Salomon, D.: Data Compression: The Complete Reference. Springer, Heidelberg (2007)
29. Sommer, R., Paxson, V.: Outside the closed world: On using machine learning for network intrusion detection. In: IEEE Symposium on Security and Privacy, pp. 305–316 (2010)
30. Song, Y., Keromytis, A.D., Stolfo, S.J.: 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. Vovk, V., Gammerman, A., Shafer, G.: Algorithmic Learning in a Random World. Springer-Verlag New York, Inc., Secaucus (2005)
32. Wagner, D., Soto, P.: Mimicry attacks on host-based intrusion detection systems. In: Proceedings of the 9th ACM Conference on Computer and Communications Security, CCS 2002, pp. 255–264. ACM, New York (2002)
33. Wang, K., Parekh, J.J., Stolfo, S.J.: Anagram: A Content Anomaly Detector Resistant to Mimicry Attack. In: Zamboni, D., Kruegel, C. (eds.) RAID 2006. LNCS, vol. 4219, pp. 226–248. Springer, Heidelberg (2006)
34. Wang, K., Stolfo, S.J.: 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)
35. Welford, B.P.: Note on a method for calculating corrected sums of squares and products. Technometrics 4(3), 419–420 (1962)
36. Wojtczuk, R.: Libnids (2011), `http://libnids.sourceforge.net/` (Online; accessed May 9, 2011)