Comparing Anomaly Detection Techniques for HTTP

Kenneth L. Ingham[1] and Hajime Inoue[2]

[1] University of New Mexico, Computer Science Department, Albuquerque, NM, 87131, USA
ingham@cs.unm.edu
[2] Carleton University School of Computer Science Ottawa, ON, K1S 5B6, Canada
hinoue@ccsl.carleton.ca

Abstract. Much data access occurs via HTTP, which is becoming a universal transport protocol. Because of this, it has become a common exploit target and several HTTP specific IDSs have been proposed as a response. However, each IDS is developed and tested independently, and direct comparisons are difficult. We describe a framework for testing IDS algorithms, and apply it to several proposed anomaly detection algorithms, testing using identical data and test environment. The results show serious limitations in all approaches, and we make predictions about requirements for successful anomaly detection approaches used to protect web servers.

Keywords: Anomaly detection, Intrusion detection, Comparison, HTTP, Hypertext transport protocol.

1 Introduction

The Hypertext Transfer Protocol (HTTP) [14] has become a universal transport protocol. For example, it is used for file sharing [19], payment processing [12], remote procedure calls [29], streaming media [1], and even protocols such as SSH [40]. Custom web applications and the rush toward Web Services [3] mean that in the future, we can expect heavier use of HTTP. Robertson et al. [32] claimed that many web applications are written by people with little expertise in security and that web-based vulnerabilities represent 25% of the total security flaws reported in the Common Vulnerabilities and Exposures list (CVE) [5] 1999 through 2005.

The importance of HTTP and the security problems have led many researchers to propose intrusion detection systems (IDSs) for use with HTTP. Unfortunately, the proposed IDSs suffer from one or more of the following problems:

- The proposed IDS is not fully described and the source code is not available.
- The test data is not available, preventing a direct comparison.
- The test data is not labeled, preventing replication.
- The test data is not representative of traffic seen today.

To address this problem, we describe a framework for comparing IDS algorithms, and we use this framework to compare several anomaly IDS algorithms under identical circumstances. This framework[1] and the attack data[2] are open source to encourage further experimentation. Under more rigorous testing, not all algorithms perform as well as the initial tests showed, and we discuss why some algorithms do better than others.

Three basic architectures of IDSs exist: signature detection, specification, and anomaly detection. We focus in this paper on anomaly detection. Signature detection systems cannot detect novel attacks, while specification systems require skills well beyond those commonly used when developing web applications. Additionally, whenever the protected program changes, the specification must be updated. Although we test only anomaly IDSs, the framework can be applied to signature and specification based algorithms as well.

The organization of the paper is as follows. The following section, Section 2, sets the stage by describing previous IDS testing, with a focus on systems designed for HTTP. We then briefly describe the test framework and test data in Section 3. The specific algorithms we tested are described in Section 3.3, followed by the test results in Section 4. Our discussion of the results follows in Section 5, while Section 6 concludes the paper with a summary of our results and a discussion of future work.

2 Prior Work

There are at least two reasons to testing IDSs: (1) to verify that an algorithm is effective and efficient at detecting attacks, and (2) to compare two or more algorithms to determine the better under various metrics.

Most IDS testing is little more than asking, "Does the IDS detect one or a few attacks?" Better are researchers who ask the question, "Which of the following attacks can the IDS detect?" Even this testing is often acknowledged as weak.

Good testing is repeatable; the data are available to other researchers facilitating direct comparisons of the results, the training data are representative of real systems, and the attack data accurately represent the diversity of attacks. A good test also compares two or more valid approaches (i.e., no straw man arguments). The results of a good test should provide guidance about which system or algorithm performs best under different circumstances. To this point, most IDSs for web servers have been weakly tested, and/or the tests are limited in their scope. In their review of IDS testing, Athanasiades et al. state that they do not believe this problem will ever be properly solved [2].

There are several explanations for the scarcity of good IDS testing. Identifying appropriate data is difficult—the data must be representative of realistic operating conditions. Data collected live from a network might be subject to

[1] The parser, framework and algorithm implementation code is available from the Comprehensive Perl Archive Network (CPAN) at `http://cpan.org/modules/by-authors/id/I/IN/INGHAM/`

[2] The attack data is available at `http://www.i-pi.com/HTTP-attacks-JoCN-2006/`

privacy concerns. Synthetic data must be shown to represent real data on a target network accurately. In order to test an IDS, researchers need a collection of intrusions and vulnerable machines on which to test the intrusions. Because a library of intrusions represents a threat to vulnerable systems, researchers often use disconnected networks for testing to ensure that the attack does not escape into unprotected networks.

Setting up and maintaining a good, protected network is resource-intensive, both in the costs of the hardware, as well as in system administration support to set up and maintain a diversity of machines needed to ensure a good test environment. Exploits are specific to operating system and version, as well to to specific compilers, libraries, and other software. An intrusion is likely to fail if any part of the execution environment is different than expected. Because of this, a machine, or virtual machine, may be required for each new intrusion added to the attack corpus.

Finally, Debar noted that a set of criteria for evaluating an IDS does not exist [11]. Even if such criteria were available, the most careful comparisons, such as Warrender et al. [39], lack enough information to be repeatable.

2.1 Frameworks for Testing

A framework for testing is one way of reproducibility by providing a setup in which different IDSs can be tested under identical conditions. Three researchers or research groups have established such frameworks:

- The first published papers about an IDS testing framework and methodology were from Puketza et al. [30,31] at UC Davis. Unless they failed to publish further work, they built the framework and then tested only one IDS: NSM [17,18].
- Wan and Yang [37] developed a framework for testing sensors that used the Internet Engineering Task Force (IETF) Intrusion Detection Working Group (IDWG) Intrusion Detection Message Exchange Format (IDMEF) [6]. Their framework might be useful, but the paper describes only a preliminary version.
- IBM Zurich [11] set up a laboratory for testing IDSs. Their normal data came not only from recordings of user sessions, but also from the IBM test suites for the AIX operating system. While this test suite is not representative of actual user interactions, it exercises normal functionality of the product.

2.2 Data Sets for Testing HTTP IDSs

Using a good data set is critical for the test. The training and test data must be representative of the web server(s) to be protected, and the attacks used for testing need to illustrate the diversity of attacks existing today. Given the diversity between web sites, the ideal situation is to use data collected from the server to be protected. These data often have privacy issues associated with them, preventing other researchers from using it and thereby hindering repeatability. This tension has resulted in some researchers using open, less-representative data, while others use closed but more accurate data sets.

The DARPA/MIT Lincoln Laboratories IDS tests of 1998 and 1999 produced the most prominent data sets [15,24]. Many researchers in IDS research used these data because large data sets are scarce and the dataset provides an immediate comparison with the original Lincoln Labs test. Open datasets allow comparison of methods, but careful analysis of the relevant papers is required to combine and compare the results. Furthermore, differences in testing methodologies make direct comparison difficult.

However, this data set is not without its critics. McHugh [27,28] pointed out that the DARPA/MIT Lincoln Laboratories IDS test used generated data, but the MIT researchers never did any tests to show that the generated data was representative of real data. Additionally, they did no tests to verify that their attacks were representative of real attacks. The Lincoln Labs data set is also quite dated, as web behavior has evolved significantly over the years.

When testing IDSs for HTTP, researchers using the Lincoln Labs data sets have only four web attacks. When systems developed using these data are tested on a broader data set, their performance suffers; confirmation of this assertion appears in this paper. In spite of these limitations, Wang and Stolfo [38], Mahoney [25], and Mahoney and Chan [26] Vargiya and Chan [36] used one or both of these data sets for testing their IDSs, at least a portion of which were for protecting web servers. Estévez-Tapiador et al. [13] used these data as normal behavior, but they developed their own attack database to supplement the attacks in the Lincoln Labs data.

Recognizing the shortcomings of the Lincoln Labs data, other researchers have used test data that is more representative for the servers the IDS is protecting. However, these data are unavailable for others to use, eliminating direct comparisons. For example, Kruegel et al. [22,23] tested their system using extensive normal data sets from multiple sites (including Google).[3] For a portion of their 12 attacks, they used attacks against software that ran on one of their data source web servers. Wang and Stolfo [38] used data collected from their departmental web server as an additional source of data, but they did not filter attacks from the data and therefore used it only for testing the training. Tombini et al. [35] collected data from two production web servers, one academic, and one industrial, with a combined total of over five million HTTP requests from web server log files. Estévez-Tapiador et al. [13] used 1500 attack requests representing variants of 85 distinct attacks, the largest attack database reported to date.

Another important HTTP data issue is how much of the HTTP request the IDS used. While most attacks to date have been in the requested resource path, some attacks target other regions of the request. For example, Apache Sioux [8] exhausts Apache's memory by a repeated header line. Wang and Stolfo [38], in different experiments, modeled the packet payload, the first and last 100 bytes,

[3] The Google data was not even available to the researchers; they sent their programs to Google, who returned the results.

and also the first 1000 bytes of the connection. Kruegel and Vigna and Kruegel et al. [22,23] obtained their test data from web server log files, and only looked at CGI programs. Web server log files are a popular data source; Tombini et al. [35] and Robertson et al. [32] also used them. Unfortunately, log files contain only a small portion of most HTTP requests, and attacks not in the resource path are unlikely to appear in the log files.

3 Experimental Setup

To perform rigorous tests of HTTP IDS algorithms, the test circumstances and data must be identical. Testing requires data representative of what production web servers receive. Quality test data is difficult to obtain; organizations with the most interesting data typically consider it confidential. Therefore, we collected data for testing from four web sites. The attack data needs to be representative of the broad range of attacks existing today. Since, as we noted in Section 2.2, no public database of attacks exists, we compiled our own. Due to space limitations, full details of the experimental setup are described by Ingham [20].

3.1 Data

The normal data set is a collection of HTTP requests received by to the University of New Mexico Computer Science departmental web server (cs.unm.edu), as well as aya.org, explorenm.com, and i-pi.com. The training data was from one week, and the normal test data is from the following week. All attacks were filtered from the data using a combination of *snort* and manual inspection. All the data sets contain the *entire* HTTP request.[4] These include information not usually found in the log files. Having the HTTP header lines allows testing for attacks not contained in the requested resource path.

The attack database contains 63 attacks, some of which are variants of the same vulnerability—either a different exploit for the same vulnerability or the same exploit against a different operating system. We include the variants because some IDS algorithms will find some variants easier to detect than others. As one example, some of the Nimda variants are short, allowing detection by the length algorithm, while others are average length.

The attacks were collected from the following sources: Attacks against web servers under test (attacks in the wild); BugTraq and the SecurityFocus archives `http://www.SecurityFocus.com/`; the Open Source Vulnerability Database http://www.osvdb.org/; the Packetstorm archives `http://Packetstorm.widexs.nl/`; and Sourcebank `http://archive.devx.com/sourcebank/`. In many cases, the attack programs from these sources contained bugs, and we had to modify the program before it would produce malicious web requests. Note that we did not test to verify whether the attacks produced could actually compromise the targeted web application.

[4] These data were captured using a *snort* filter which reconstructs the application layer portion.

The attack database contains the following categories of attacks: buffer overflow; input validation error (other than buffer overflow); signed interpretation of unsigned value; and URL decoding error. The attacks targeted different web servers: Active Perl ISAPI; AltaVista Search Engine; AnalogX SimpleServer; Apache with and without mod_php; CERN 3.0A; FrontPage Personal Web Server; Hughes Technologies Mini SQL; InetServ 3.0; Microsoft IIS; NCSA; Netscape FastTrack 2.01a; Nortel Contivity Extranet Switches; OmniHTTPd; and PlusMail. The target operating systems for the attacks include the following: AIX; Linux (many varieties); Mac OS X; Microsoft Windows; OpenBSD; SCO UnixWare; Solaris x86; Unix; VxWorks; and any x86 BSD variant.

3.2 The Algorithm Test Framework

A framework allows testing a collection of algorithms in the same environment, ensuring that each algorithm is working under identical conditions. By providing a common interface, testing any IDS algorithm that uses this interface is straightforward, and the surrounding support code is reused. The framework for running the tests was designed to work with anomaly detection algorithms, but it is general enough to work with signature and specification systems—these systems simply need no training before testing. As an example, it was easy to write an IDS algorithm object to use *snort* signatures for HTTP requests. Detailed descriptions of the test framework are available in [20].

Some algorithms require that the data be tokenized. For these algorithms, we implemented a parser that breaks the HTTP request into tokens based on the those specified in the HTTP standard, RFC 2616 [14]. The tokens are a combination of the token type (e.g., `method`) and optionally the value (e.g., `GET`). In practice, most of the values are necessary to properly distinguish attacks from normal requests. The result is a stream of tokens combined with the associated values.

Instead of using tokens, some algorithms use a string representation for the request. This (much simpler) representation is also available from the parser.

3.3 Algorithms

We consider algorithms from Kruegel and Vigna [22], who developed a linear combination of six measures (length, a character distribution measure, a Markov Model, presence/absence of parameters, order of parameters, and whether parameter values were enumerated or random), and applied them to CGI parameters. For some of the six algorithms, we also consider them in isolation. We also implemented the character distribution metric described by Wang and Stolfo [38], and the DFA induction and n-grams described by Ingham et al. and Ingham [21,20].

These algorithms are either proposed by often cited papers in the IDS community, similar to those algorithms but using different data or representations, or successful in related domains. In short, we tested algorithms claimed to be or likely to be successful in HTTP-based anomaly intrusion detection.

Request Length. Observing that buffer overflows and cross-site scripting attacks tend to be longer than normal CGI attribute values, one measure used by Kruegel and Vigna [22] was the mean μ and variance σ^2 of attribute lengths. These values were calculated from training data.

For testing, the system calculated the probability p that an attribute would have the observed length l by:

$$p = \frac{\sigma^2}{(l-\mu)^2}$$

Character Distributions. Buffer-overflow attacks often have a distinctive character distribution. Two research groups have compared the character distribution of test instances to the distribution in the training data. Wang and Stolfo [38] used a character distribution metric on similarly-sized packets. Kruegel and Vigna [22] used a character distribution as one of six tests.

Mahalanobis distance. Wang and Stolfo [38] measured the Mahalanobis distance, d, between two distributions. For efficiency reasons they used a measure they called the *simplified Mahalanobis distance*:

$$d(x, \overline{y}) = \sum_{i=0}^{n-1} \frac{\mid x_i - \overline{y}_i \mid}{\overline{\sigma_i} + \alpha} < \infty$$

n is 256 for the ASCII character set. The α term is a smoothing factor so that the distance does not become infinite when $\overline{\sigma_i}$ is 0. Wang and Stolfo did not specify how they calculate α; for the results reported in this paper, $\alpha = 0.001$. Wang and Stolfo set the distance threshold to 256 (one standard deviation). Using this value means that rare distributions are anomalous; consequently it reports false positives even when tested on the training data set.

Our implementation differs with Wang and Stolfo's slightly. They correlated packet length with character frequencies. Our data consists only of the data at the application layer; the raw packets containing the data were not stored. Therefore, we apply this method to the complete request. Note that while the different packet sizes may have a given character distribution, an attacker can easily control the packet size, allowing them to use packets of a size with a better match for the character distribution.

χ^2 of idealized character distribution. As one of six tests, Kruegel and Vigna [22] use a measure of relative character frequency. They produced a sorted list of character frequencies f_c containing the relative frequency of the character c. Their example is the string `passwd`, where the absolute frequency distribution is 2 for `s`, 1 for `a`, `d`, `p`, and `w`, and 0 for all other characters. The relative frequencies are then $f = (\frac{1}{3}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, \frac{1}{6}, 0, ..., 0)$; note that f_6 through f_{256} are 0. Kruegel and Vigna noted that relative frequencies decrease slowly for non-attack requests, but have a much steeper decline for buffer overflows, and no decline for random data.

They called the character distribution induced from the training data the *idealized character distribution* ($\mathcal{ICD}$) and noted that $\sum_{i=1}^{256} \mathcal{ICD}(i) = 1.0$. As mentioned in the prior paragraph, the $\mathcal{ICD}$ is sorted so most common frequency is $\mathcal{ICD}(1)$ and the least common is $\mathcal{ICD}(256)$. $\mathcal{ICD}$ is calculated during training as the average over the character distributions of the requests in the training data.

For testing, they binned the $\mathcal{ICD}$ (the expected distribution, calculated through training) and the distribution of the test request (observed distribution) into six bins as follows:

Bin	1	2	3	4	5	6
i	1	2–4	5–7	8–12	13–16	17-256

where $i \in [1, 256]$. For example, bin 4 contains $\sum_{i=8}^{12} ICD(i)$. Once binned, they then use a χ^2 test to determine if the character distribution of CGI parameter values is similar to that of the training data:

$$\chi^2 = \sum_{i=1}^{6} \frac{(O_i - E_i)^2}{E_i}$$

where E_i is bin i for the $\mathcal{ICD}$, and O_i is bin i for the observed distribution. χ^2 is compared to values from a table and the corresponding probability is the return value.

CGI Parameter Measures. Kruegel and Vigna [22] used three different observations about CGI parameters. First, they noted that since CGI parameters are set programmatically, the normal order of the parameters is fixed. If a human generates the path, the order could be different, and they presumed this change indicated a potential attack. For similar reasons, they also noted CGI parameters are supplied even when they have no value. The result is a regularity in the number, name, and order of the parameters. Their system learned the parameters present for a given CGI program path. When testing an instance, the return value is 1 if the same parameters appeared in the training data as in the test instance, and 0 otherwise.

Similar to the presence and absence test, Kruegel and Vigna noted that some CGI parameter values are selected from a finite set (enumerated), and others are effectively random. In the training phase, they test to see whether the number of parameter values stays small compared to the number of examples. If it does, then the parameter values are enumerated and the algorithm performs no generalization. Otherwise, it accepts any value during testing.

DFA. We use a one-pass, $O(nm)$ DFA induction algorithm where n is the number of samples in the training data set and m is the average number of tokens per sample. The algorithm does not require negative examples. This algorithm is described in detail by Ingham et al. [21].

A DFA by itself is simply a language acceptor; however, we expect some variation in normal behavior not incorporated in the DFA induction algorithm.

When testing, the algorithm notes when it is unable to make a transition on a token. If a state exists which is a destination of that token, the DFA is adjusted to that state. If not, the algorithm uses the next token and tries again. The number of missed tokens is used to calculate the similarity s between the DFA model and an HTTP request:

$$s = \frac{\#\text{ of tokens reached by valid transitions}}{\#\text{ of tokens in the HTTP request}} \in [0,1]$$

The similarity measure reflects the proportion of the request requiring changes for the DFA to accept the request. Using proportionality instead of a raw miss count allows complex requests to have greater variability than simpler ones.

Markov Model. A Markov model is a nondeterministic finite automaton (NFA) with probabilities associated with the transitions. A Markov model differs from a DFA in that multiple transitions might exist for a given token, and a probability is associated with each transition. The probability of a given string of tokens can be calculated as the sum of the probabilities of each independent path through the NFA that can generate the string of tokens. The probability of a given path is the product of the probabilities of each of the transitions, and this probability is interpreted as the similarity measure for the testing. Similar to a DFA, a Markov Model represents the structure of the HTTP request through a directed graph.

For an anomaly detection system, the traditional approach is to build an NFA that exactly matches the training data. Through a series of state merging operations, it is compressed and hence it becomes more general (and, as a side effect, it becomes a DFA with probabilities). For more details about Markov model induction, see the work by Stolcke [34] and Stolcke and Omohundro [33]. Warrender et al. noted that building a generalized Markov model is $\mathcal{O}(n^2)$ [39].

Markov models have been shown to be an effective but time-consuming algorithm for system-call based intrusion detection [39]. Kruegel and Vigna [22] used a Markov model as a portion of the IDS for protecting web servers, but after noting that the probability of any given request string is small, they used their Markov model as a DFA, noting only whether or not the model was capable of generating the string in question.

Our Markov model implementation is a modification of the DFA algorithm described in Section 3.3. When learning the DFA, the number of times that a transition is taken is recorded, and the probability of taking a given transition is the fraction of the sum of all of the transitions that the taken transition represents. This approach is not exactly the same as a more traditional Markov model, but the result is similar in size and effect to a Markov model after generalization.

Linear Combination. Combining IDSs is a logical step once more than one IDS is available. The system developed by Kruegel and Vigna [22] was limited to HTTP CGI requests, and consisted of a linear combination of the **length**, **character distribution**, **order**, and **presence or absence** of CGI parameter values. Additionally, it also included a test for which CGI parameter values were **enumerated or random**, and a **Markov model** to learn the structure of those values.

The threshold for normal for each algorithm was determined dynamically, chosen to be 10% above the highest value obtained in training. Calculating the threshold requires a second pass over the training data, testing it to find the maximum value for each measure. For testing, each algorithm was equally weighted and the system produced a binary normal/abnormal signal.

***n*-grams.** An n-gram [9] is a substring generated by sliding a window of length n across a string of tokens. The result is a set of strings of length n. For example, given the string `abcdef` and $n = 3$, the resulting 3-grams are: `abc`, `bcd`, `cde`, and `def`. The similarity measure considers the presence or absence of the test n-grams in the set of n-grams learned from the training data:

$$s = \frac{\#\text{ of } n\text{-grams from the request also in the training data}}{\#\text{ of } n\text{-grams in the HTTP request}} \in [0, 1]$$

The n-gram algorithm can use either tokens or strings from the data source. Early testing showed poor results for strings, so we report results using tokens as the alphabet.

Targeted Generalization Heuristics. To improve the accuracy of the n-gram and DFA induction algorithms, we also applied several heuristics that increase the generalization. These check that certain data types have a valid (parsable) format. If so, they return a small, enumerated set of values dependent on the heuristic. The data types that are checked for valid form are host names, IP addresses, dates, various hash values (PHP session IDs, HTTP entity tags, etc), floating point numbers (HTTP q-values), and email addresses. Ingham and Ingham et al. provide a detailed description of these heuristics in [20,21].

4 Results

The traditional method for reporting IDS results is a receiver operating characteristic (ROC) curve that shows the tradeoff between identifying real attacks (true positives) and incorrectly flagging non-attack requests as an attack (false positives) [16]. True or false positives are represented in the ROC curves presented here as the fraction of the attack database or test data set properly or improperly identified. Each set of connected points represents a different data set used with the algorithm, and each point represents a different similarity threshold for distinguishing normal from abnormal. A perfect algorithm would have a point at $(0, 1)$ (the upper-left corner of the graph) indicating no false positives and 100% correct identification of attacks. In order to better see the most accurate range, the plots only show the X axis values in $[0, 0.1]$. The portion of the plot in the rest of the X axis represents a range where the false positives would be too high for production use; we visit this claim in Section 4.7. The axes in these plots indicate the actual fraction of true and false positives in the test.

To ease comparisons between algorithms, most of the ROC plots have the same scale; one required a different scale to present the data, and this fact is noted in the plot description.

McHugh noted several potential problems in presenting IDS test results with ROC curves [28]. His first objection is that some researchers presented curves with only one measured point and assumed continuity from (0,0), through their point, to (1,1). We present plots with 128 uniformly divided points in $[0, 1]$. No assumption is made about (0,0) or (1,1). McHugh also pointed out that for the ROC curves to be comparable, the unit of analysis must be the same. For every test in this paper, this unit of analysis is always one HTTP request. The tests we performed used the data and framework described in Section 3.

4.1 Length

Accuracy is below 80% true positive at tolerable false positive rates (see Figure 1). This measure can detect some buffer overflows and cross-site scripting attacks, however, attacks such as the Apache chunked transfer error [4] and some variants of Nimda [10] are short enough to pass as normal; if they are too short, padding to increase the length is easy. Therefore, a minimum length will never stop an attack other than by a simplistic attacker. Because this algorithm accepts many strings that are not legal HTTP, an attacker has great freedom in the construction of her attack.

If this algorithm were to be applied to tokens, it would overgeneralize. Consider how many sentences with n words are valid English-language sentences. Therefore, this algorithm is unlikely to ever be useful in isolation. It might be applied as one of several algorithms, assuming non-attack requests have a tight enough upper bound on their length.

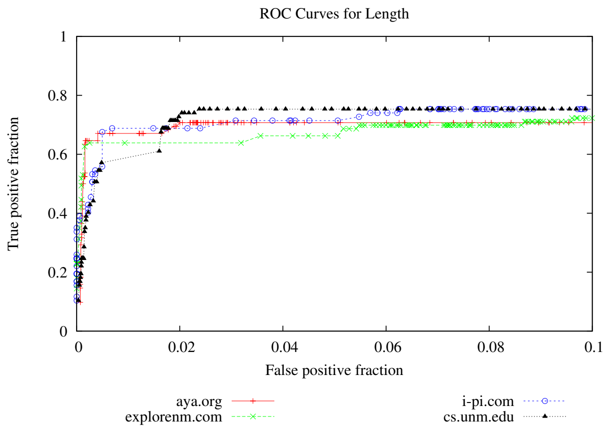

Fig. 1. Receiver Operating Characteristic curves showing the accuracy of the length algorithm

4.2 Character Distributions

Our Mahalanobis distance results (see Figure 2) differ from Wang and Stolfo's [38]. Note that the cs.unm.edu accuracy is lower than other sites, indicating that the measure's accuracy depends on the mix of HTTP requests. Wang and Stolfo reported true positive rates about 90% with a 20% false positive rate on the Lincoln Labs data. Trained and tested using their own departmental server, the false positive rate improved, ranging from 0.0084% to 1.3%. They found their system did not always detect variants of exploits used during training. A possible explanation is their dependence on packet size in their calculations. As we noted in Section 3.3, an attacker can easily manipulate packet size, so we question the usefulness of this correlation.

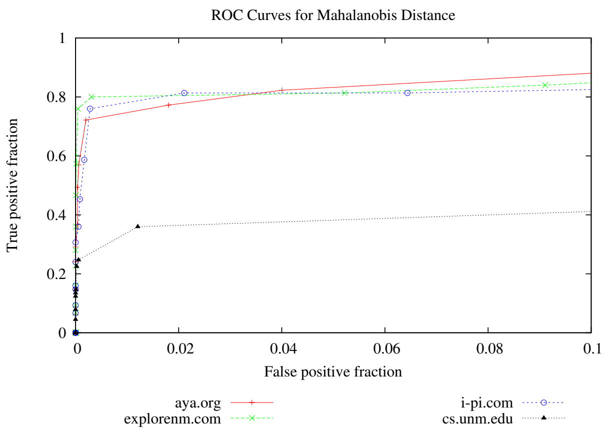

Fig. 2. Receiver Operating Characteristic curves showing the accuracy of the Mahalanobis distance algorithm

Figure 3 contains the χ^2 distance results. This algorithm performs poorly on all data sets, with a true positive rate at or below 40%.

The Mahalanobis distance and χ^2 distance algorithms generalize by allowing similar, instead of identical, character distributions. Unfortunately, this approach fails. The HTTP protocol is flexible enough that an attack can be padded to give a character distribution considered close enough to normal, especially with the myriad ways of encoding data allowed by the standards. To make the problem worse for these metrics, some attacks such as the Apache chunked transfer error [4] and some variants of Nimda [10] use a character distribution that might pass as normal without padding, and had the attacker needed to, she could have easily made minor changes to the attack (such as putting the proper host name or IP address in the `Host:` field) as needed to ensure a valid character distribution. The problem is that the set considered normal is so large that it includes many of the attacks in the attack database, regardless of if the attack is legal HTTP or not.

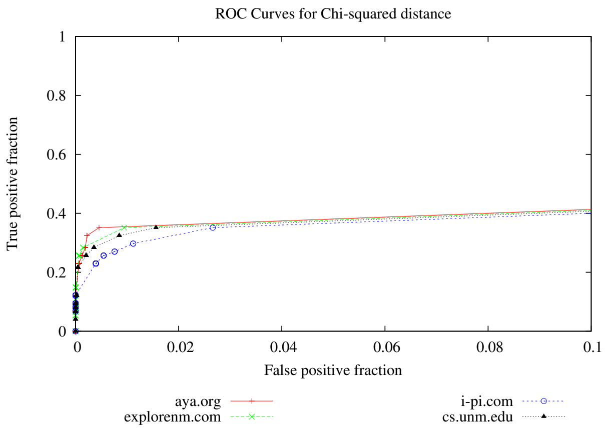

Fig. 3. Receiver Operating Characteristic curves showing the accuracy of the χ^2 distance algorithm

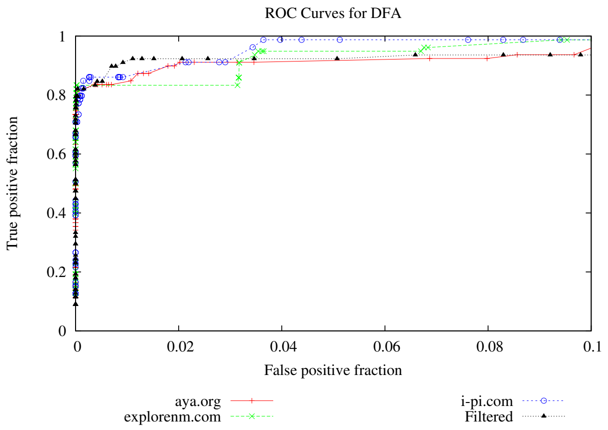

Fig. 4. Receiver Operating Characteristic curves showing the accuracy of the DFA algorithm

Wang and Stolfo [38] tested the Mahalanobis distance using the MIT Lincoln Labs data (Section 2.2). This data set contains only four HTTP attacks. In the years since the MIT data were collected, attack characteristics have changed; our more comprehensive attack data set illustrates the effect of this difference on this algorithm (Figure 2).

4.3 DFA

Figure 4 shows the DFA accuracy. The DFA can achieve better than 80% true positive rate at a false positive rate of less than 0.1%, which is better than all but the 6-grams. At slightly higher false positive rates, it achieves true positive rates of over 90%.

The DFA induced using tokens is a directed graph representing the structure of the HTTP request. Generalization occurs in the DFA generation described in [21]. It also occurs when one or more "missed tokens" are allowed. These generalizations are limited compared to that performed by the length and character distribution algorithms. The better true positive rate relative to all of the other algorithms shows that the model is even more accurate than that of the n-grams.

4.4 Markov Model

The Markov model result values are in $[0, 10^{-13}]$ with many values as small as 10^{-300}. These small values make it appear that the algorithm identifies everything (both normal traffic and attacks) as abnormal. To better understand these results, Figure 5 shows the data plot where the similarity value from the Markov model m has been transformed into a new similarity value s by $s = \frac{1}{|\log_e(m)|}$, and the plot scale has been changed so the data appears (making these plots not directly comparable to the rest of the ROC plots in this paper). This transformation means that the data cannot go through the point $(0, 0)$, and all of the data appears on the plot. The log transformed Markov model provides 94% accuracy on cs.unm.edu data, but with an unacceptable false positive rate. The results on the other web sites show an even better true positive rate, but the false positive rate remains unacceptably high.

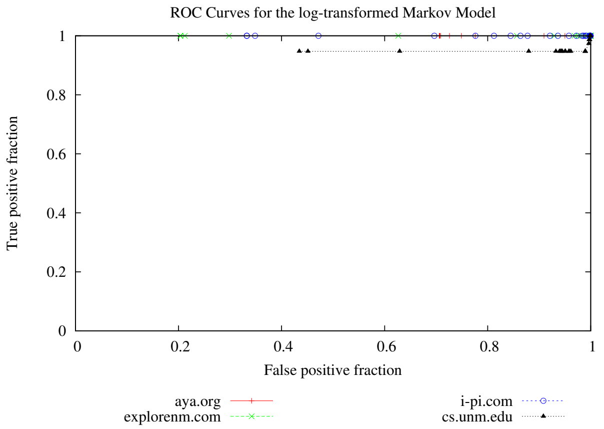

Fig. 5. Receiver Operating Characteristic curves showing the accuracy of the Markov model algorithm. Note that the scale on this plot does not match the scale of the other plots.

In a Markov model, normal requests might have a probability of 0 due to minor differences from the instances in the training data. If the model was induced from filtered data, attacks would also result in a probability of 0, and the model has a hard time distinguishing between these two cases. The Markov model's generalization is traditionally achieved by allowing probabilities within a given range. The diversity of normal requests means any given normal request is unlikely, and perpetual novelty of HTTP data leads to normal requests with a probability of 0. The combination of these two factors means that the Markov model is a poor model for HTTP requests. Our results applying a Markov model to the tokens of the complete HTTP request using tokens mirror those of Kruegel and Vigna applying it to CGI parameters [22]. They reported that the Markov model suffered because HTTP requests are so diverse that the probability of any given request is low. When working with complete requests, the problem is even worse, because the increased number of tokens increases the normal level of diversity, resulting in lower probabilities for any given HTTP request.

4.5 Linear Combination

The linear combination results are in Figure 6. The accuracy is best on the cs.unm.edu data, but the true positive rate is only around 60%. On the other web sites, it is less accurate. Kruegel and Vigna reported a true positive rate of 100% and false positive rates less than 0.000650. The disparity is explained by the attacks attempted—in contrast to their attack database which was constructed solely of attacks in CGI parameters, these attacks account for only 40% of the attacks in our database.

Most of the discrimination the linear combination came from the **order**, **presence or absence**, and **enumerated or random** tests which did not generalize. We found it was not hampered by the character distribution overgeneralization because they limited their work to a small portion of all attacks (CGI parameters) and this measure was but one of six.

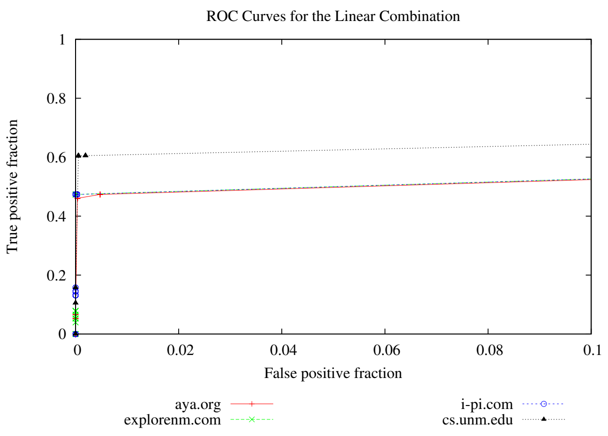

Fig. 6. Receiver Operating Characteristic curves showing the accuracy of the linear combination algorithm

The method of combining IDSs itself determines the generalization of the combined algorithm. If all models must agree that a request is normal, the least general usually determines a request is abnormal. Combining overgeneralizing detectors such as length and character distribution will usually indicate a normal request (including for many attacks), and therefore contribute little to the discrimination power of the combination; combining overgeneralizing detectors results in a system that overgeneralizes.

4.6 *n*-Grams

Results for 6-grams[5] are in Figure 7. The accuracy starts at around 85% true positive rate with a low false positive rate, making this algorithm comparable to the DFA for accuracy.

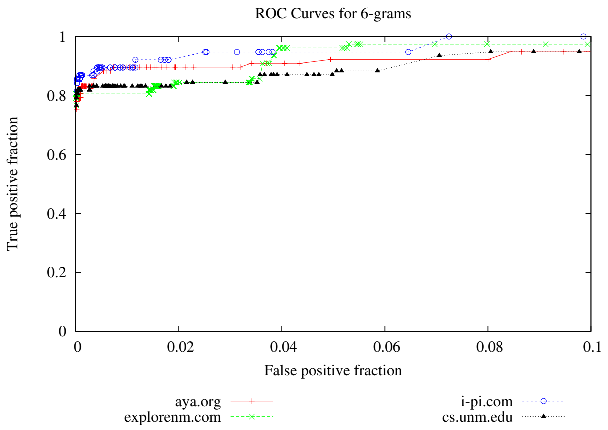

Fig. 7. Receiver Operating Characteristic curves showing the accuracy of the 6-gram algorithm

n-grams effectively model the structure, or grammar, of a request by encoding sequences of tokens as a directed graph in a manner similar to the DFA and Markov Model. *n*-grams minimize false positives by allowing a small number of mismatches, and it is better able to tell normal requests from nonsense.

4.7 False Positives

A human system administrator would have to inspect false positives to determine if they represent normal traffic or attacks. When comparing the algorithms, a useful metric is the load that the algorithm would place on this person. Table 1 shows the false positive rate per day, assuming a true positive rate of only 80% is required. This table presents data using the cs.unm.edu data sets and shows that only the 6-grams and the DFA have a false positive rate that might be acceptable for a web site like the UNM CS department.

Most previous research has reported false positives as the fraction of the non-attack test data misidentified, which is the value shown in the ROC plots presented in the earlier sections. This result can be misleading for web sites if a

[5] Preliminary tests showed $n = 6$ to be optimal.

human must evaluate the abnormal requests to determine if if they represent attacks. A 1% false positive rate on a lightly-visited web site may be tolerable; the same percentage on Amazon.com or Google.com would require a large full-time staff. A false positive rate of 0.01 corresponds to 917, 50, 8, and 43 false positives per day for cs.unm.edu, aya.org, i-pi.com, and explorenm.com respectively. In the 1999 DARPA/MIT Lincoln Laboratories IDS tests, they stated that above 10 false positives per day is a high rate [15].

Table 1. False positive rate per day for the algorithms, trained and tested using the cs.unm.edu data. Algorithms marked with ∞ did not achieve a threshold 80% true positive rate. The Markov Model data transform is described in Section 4.4.

Algorithm	**FP/day**
Mahalanobis distance	91,524
χ^2 of $\mathcal{ICD}$	∞
Length	∞
6-grams	13
DFA	37
Markov Model (log transform)	39,824
Linear combination	∞

5 Discussion

The results in Section 4 show that character-based algorithms (Mahalanobis distance, χ^2 of $\mathcal{ICD}$, and Length) are notably less accurate than two of the token-based algorithms (DFA, n-grams). Tokens represent a higher lexical unit, and are used by the system to represent meaning; attacks often represent nonsensical requests. With the need to "ship the product yesterday" and other deadlines, programmers often focus on making the system work under common circumstances and spend fewer resources on exceptional cases. Additionally, to consider all of the ways in which exceptional states may be represented requires thinking in ways many programmers were not trained. In our attack database, most, if not all, of the attacks are nonsensical. The ability to represent more of the meaning of a HTTP request improves the ability of an algorithm to discriminate between normal and abnormal. Presumably the normal requests do not represent nonsense. Applying these concepts to the algorithms we tested, the DFA and n-grams learn the higher-level structure of valid HTTP requests, and so therefore they can use this structure to better tell if a request is normal or not.

Both token based algorithms share a weakness in their similarity measures in that they cannot discern between a novel request with a few new tokens and an attack with a small number of tokens. This was responsible for some of the missed attacks. Another attack missed by the DFA can be traced to a user typo.

The pair of tokens `//` appeared in the training data, causing an edge from the node corresponding to the path separator `/` back to itself. Unfortunately, the beck attack [7] used a multitude of `/`s to cause an out-of-memory condition in an older version of Apache.

The idea of representing the meaning of the request allows us to make a prediction: Statistics such as character distribution applied to tokens rather than characters may be more accurate than when the same statistic is applied to the characters making up the request. However, the relationships between tokens is important to the semantics. Statistics on tokens are likely to be less accurate unless the measure can represent these relationships. In effect, by ignoring the relationships between tokens, measures such as the character distribution algorithms applied to tokens will continue to overgeneralize, and therefore be more prone to mimicry attacks. Consider as an example all English-language sentences with a specific distribution of words versus the sentences that are well-formed and not nonsense.

6 Conclusion

This paper evaluated and compared seven different different anomaly intrusion detection algorithms for HTTP under realistic conditions. This testing is more rigorous than any HTTP IDS testing reported to date. For this comparison we implemented an open-source IDS testing framework. In addition, we developed the most comprehensive open database of HTTP attacks designed for IDS testing.

Most previous IDS approaches for HTTP have represented the request as a character string. The work we report is one of the first to use tokens from parsing the request, and the first to use these tokens with DFA induction and n-grams. These algorithms detect more attacks than earlier approaches. One reason for this improved accuracy is that we use the complete HTTP request instead of just a portion—most previous IDSs ignore portions of the request and obviously cannot detect attacks in the ignored portions.

Our test results are explained by two factors. The first is the data representation of the HTTP request. We have shown that the token-based methods result in algorithms with a better ability to discriminate between sense and nonsense, and as a result, between legitimate requests and attacks. The second factor is generalization. We included several heuristics for generalization the algorithms using tokens. A detailed discussion of the effects of generalization is out of scope for this paper but is provided by Ingham in [20].

This research has shown that all the algorithms have an unacceptable false positive rate. We need additional algorithms and heuristics to improve performance. Furthermore, our work implies that new approaches should be token based, because they better represent HTTP requests than current algorithms. We hope that the IDS testing framework described in this paper encourages further research.

Acknowledgments

We thank the anonymous reviewers for their comments and suggestions. The first author was partially supported by National Science Foundation grant ANIR-9986555. The second author is supported by the Canadian Government through MITACS.

References

1. Apple Computer: Tunneling RTSP and RTP over HTTP (2006) (accessed September 13, 2006), `http://developer.apple.com/ documentation/QuickTime/QTSS/Concepts/chapter_2_section_14.html`
2. Athanasiades, N., Abler, R., Levine, J., Owen, H., Riley, G.: Intrusion detection testing and benchmarking methodologies. In: IEEE-IWIA '03: Proceedings of the First IEEE International Workshop on Information Assurance (IWIA'03), Washington, DC, USA, page 63, IEEE Computer Society, Los Alamitos (2003)
3. Booth, D., Haas, H., McCabe, F., Newcomer, E., Champion, M., Ferris, C., Orchard, D.: Web services architecture. Technical Report W3C Working Group Note 11 February 2004, World Wide Web Consortium (W3C) (2004) (accessed 2007-04-05), online at `http://www.w3.org/TR/ws-arch/`
4. Cohen, C.F.: CERT advisory CA-2002-17 Apache web server chunk handling vulnerability (July 2002) (accessed July 24, 2002), `http://www.cert.org/advisories/CA-2002-17.html`
5. Corporation, M.: Common vulnerabilities and exposures (accessed June 16, 2006), `http://cve.mitre.org/`
6. Curry, D., Debar, H.: Intrusion detection message exchange format data model and extensible markup language (XML) document type definition (December 2002) (accessed January 1, 2003), `http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-09.txt`
7. cve.mitre.org: CVE-1999-0107 (July 1999) (accessed September 3, 2006), `http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0107`
8. cve.mitre.org: CVE-1999-1199 (September 2004) (accessed October 30, 2005), `http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1199`
9. Damashek, M.: Gauging similarity with n-grams: language-independent categorization of text. Science 267(5199), 843–848 (1995)
10. Danyliw, R., Dougherty, C., Householder, A., Ruefle, R.: CERT advisory CA-2001-26 Nimda worm (September 2001), `http://www.cert.org/advisories/CA-2001-26.html`
11. Debar, H., Dacier, M., Wespi, A., Lampart, S.: An experimentation workbench for intrusion detection systems. Technical Report RZ 6519, IBM Research Division, Zurich Research Laboratory, 8803 Rüuschlikon, Switzerland (September 1998)
12. Eastlake, D., Khare, R., Miller, J.: Selecting payment mechanisms over HTTP (2006) (accessed September 13, 2006), `http://www.w3.org/TR/WD-jepi-uppflow-970106`
13. Estévez-Tapiador, J.M., García-Teodoro, P., Díaz-Verdejo, J.E.: Measuring normality in http traffic for anomaly-based intrusion detection. Journal of Computer Networks 45(2), 175–193 (2004)
14. Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., Berners-Lee, T.: Hypertext transfer protocol—HTTP/1.1. RFC 2616 (June 1999) (accessed October 2, 2002), `ftp://ftp.isi.edu/in-notes/rfc2616.txt`

15. Haines, J.W., Lippmann, R.P., Fried, D.J., Tran, E., Boswell, S., Zissman, M.A.: 1999 DARPA intrusion detection system evaluation: Design and procedures. Technical Report TR-1062, Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, MA, USA (February 2001)
16. Hancock, J., Wintz, P.: Signal Detection Theory. McGraw-Hill, New York (1966)
17. Heberlein, L.: Network security monitor (NSM)—final report. Technical report, University of California at Davis Computer Security Lab, Lawrence Livermore National Laboratory project deliverable (1995), `http://seclab.cs.ucdavis.edu/papers/NSM-final.pdf`
18. Heberlein, L., Dias, G., Levitt, K., Mukherjee, B., Wood, J., Wolber, D.: A network security monitor. In: 1990 IEEE Computer Society Symposium on Research in Security and Privacy, Oakland, CA, USA, May 7–9, 1990, pp. 296–304. IEEE Computer Society Press, Los Alamitos, CA, USA (1990)
19. Hernández, L.O., Pegah, M.: WebDAV: what it is, what it does, why you need it. In: SIGUCCS '03: Proceedings of the 31st annual ACM SIGUCCS conference on User services, New York, NY, USA, pp. 249–254. ACM Press, New York (2003)
20. Ingham, K.L.: Anomaly Detection for HTTP Intrusion Detection: Algorithm Comparisons and the Effect of Generalization on Accuracy. PhD thesis, Department of Computer Science, University of New Mexico, Albuquerque, NM, 87131 (2007)
21. 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)
22. Kruegel, C., Vigna, G.: Anomaly detection of web-based attacks. In: Proceedings of the 10th ACM conference on Computer and communications security, pp. 251–261. ACM Press, New York (2003)
23. Kruegel, C., Vigna, G., Robertson, W.: A multi-model approach to the detection of web-based attacks. Computer Networks 48(5), 717–738 (2005)
24. Lippmann, R., Haines, J., Fried, D., Korba, J., Das, K.: The 1999 DARPA off-line intrusion detection evaluation. Computer Networks 34(4), 579–595 (2000)
25. Mahoney, M.V.: Network traffic anomaly detection based on packet bytes. In: Proceedings of the 2003 ACM Symposium on Applied computing, pp. 346–350. ACM Press, New York (2003)
26. Mahoney, M.V., Chan, P.K.: Learning nonstationary models of normal network traffic for detecting novel attacks. In: Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining, pp. 376–385. ACM Press, New York (2002)
27. McHugh, J.: The 1998 Lincoln Laboratory IDS evaluation—a critique. In: Debar, H., Mé, L., Wu, S.F. (eds.) RAID 2000. LNCS, vol. 1907, pp. 145–161. Springer, Heidelberg (2000)
28. McHugh, J.: Testing intrusion detection systems: a critique of the 1998 and 1999 DARPA intrusion detection system evaluations as performed by Lincoln Laboratory. ACM Transactions on Information and Systems Security 3(4), 262–294 (2000)
29. Microsoft Corporation: Exchange server 2003 RPC over HTTP deployment scenarios (2006) (accessed Sept 13, 2006), `http://www.microsoft.com/technet prodtechnol/exchange/2003/library/ex2k 3rpc.mspx`
30. Puketza, N., Chung, M., Olsson, R., Mukherjee, B.: A software platform for testing intrusion detection systems. IEEE Software 14(5), 43–51 (1997)
31. Puketza, N.J., Zhang, K., Chung, M., Mukherjee, B., Olsson, R.A.: A methodology for testing intrusion detection systems. IEEE Transactions on Software Engineering 22(10), 719–729 (1996)

32. Robertson, W., Vigna, G., Kruegel, C., Kemmerer, R.A.: Using generalization and characterization techniques in the anomaly-based detection of web attacks. In: Network and Distributed System Security Symposium Conference Proceedings: 2006. Internet Society (2006) (accessed February 12, 2006), `http://www.isoc.org/isoc/conferences/ndss/06/proceedings/html/2006/papers/anomaly_signatures.pdf`
33. Stolcke, A., Omohundro, S.: Hidden Markov Model induction by bayesian model merging. In: Hanson, S.J., Cowan, J.D., Giles, C.L. (eds.) Advances in Neural Information Processing Systems, vol. 5, pp. 11–18. Morgan Kaufmann, San Mateo, CA (1993)
34. Stolcke, A., Omohundro, S.M.: Best-first model merging for hidden Markov model induction. Technical Report TR-94-003, International Computer Science Institute, 1947 Center Street, Suite 600, Berkeley, CA, 94704-1198 (1994)
35. Tombini, E., Debar, H., Mé, L., Ducassé, M.: A serial combination of anomaly and misuse IDSes applied to HTTP traffic. In: 20th Annual Computer Security Applications Conference (2004)
36. Vargiya, R., Chan, P.: Boundary detection in tokenizing network application payload for anomaly detection. In: Proceedings of the ICDM Workshop on Data Mining for Computer Security (DMSEC). Workshop held in conjunction with The Third IEEE International Conference on Data Mining, November 2003, pp. 50–59 (2003) (accessed April 5, 2006), available at `http://www.cs.fit.edu/~pkc/dmsec03/dmsec03notes.pdf`
37. Wan, T., Yang, X.D.: IntruDetector: a software platform for testing network intrusion detection algorithms. In: Seventeenth Annual Computer Security Applications Conference, New Orleans, LA, USA, December 10–14, 2001, IEEE Computer Society, Los Alamitos, CA, USA (2001)
38. 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)
39. Warrender, C., Forrest, S., Pearlmutter, B.A.: Detecting intrusions using system calls: Alternative data models. In: IEEE Symposium on Security and Privacy, pp. 133–145. IEEE Computer Society Press, Los Alamitos (1999)
40. Wiers, D.: Tunneling SSH over HTTP(S) (2006) (accessed September 13, 2006), `http://dag.wieers.com/howto/ssh-http-tunneling/`