5-1-2007

Anomaly detection for HTTP intrusion detection : algorithm comparisons and the effect of generalization on accuracy

Kenneth Ingham III

Anomaly Detection for HTTP Intrusion Detection: Algorithm Comparisons and the Effect of Generalization on Accuracy

by

Kenneth LeRoy Ingham III

B.S. *cum laude*, University of New Mexico, 1985
M.S., University of New Mexico, 1994

DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

May 2007

Acknowledgments

A Ph.D. is never accomplished alone. Many people have supported, advised, or otherwise helped me with the long, difficult process of learning I have been on.

My advisor, Stephanie Forrest, has an ability to immediately see the overall importance of ideas, putting them into global perspective. Someday, I hope to have a fraction of this skill. Her high standards for research forced me to do better work than I would have thought I was capable of.

My dissertation committee also provided guidance and support: Barney Maccabe and Terran Lane. Anil Somayaji deserves special mention because his advice and assistance went far beyond that of an external committee member. Anil, I really appreciate your ability to see the difference between what I had done and what was needed, and then clearly articulating this difference so I could understand what was needed and why it was important.

Several people who read part or all of my dissertation, and my thanks go out to them for their sharp eyes and informative comments: James Hardy and Gabriela Barrantes. Michael Wester deserves special mention here also for his guidance and advice on the analysis of generalization set growth rates. He also understands the travails of getting a Ph.D. and his advice and support were valuable beyond words.

Partial funding for this research was provided by the National Science Foundation grant ANIR-9986555, for which I am grateful; it allowed me to focus more on my research.

Rambo, Squeaker, Hoover, and Wooly Bear also get credit for helping me smile and to see that there was more to life than Ph.D. work. Maybe we can go outside and play more now.

I could not have done all of this work without the support of my wife, Diana Northup. She understood the Ph.D. process, having received her doctorate in Biology a few years ago. She was responsible for an enormous amount of encouragement and kind words, especially when I was frustrated over my perceived lack of progress. She also provided the gentle pushes I often needed to continue to make progress.

Anomaly Detection for HTTP Intrusion Detection: Algorithm Comparisons and the Effect of Generalization on Accuracy

by

Kenneth LeRoy Ingham III

ABSTRACT OF DISSERTATION

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Doctor of Philosophy
Computer Science

The University of New Mexico

Albuquerque, New Mexico

May 2007

Anomaly Detection for HTTP Intrusion Detection: Algorithm Comparisons and the Effect of Generalization on Accuracy

by

Kenneth LeRoy Ingham III

B.S. *cum laude*, University of New Mexico, 1985
M.S., University of New Mexico, 1994

Ph.D., Computer Science, University of New Mexico, 2007

Abstract

Network servers are vulnerable to attack, and this state of affairs shows no sign of abating. Therefore security measures to protect vulnerable software is an important part of keeping systems secure. Anomaly detection systems have the potential to improve the state of affairs, because they can independently learn a model of normal behavior from a set of training data, and then use the model to detect novel attacks. In most cases, this model represents more instances than were in the training data set—such generalization is necessary for accurate anomaly detection.

This dissertation describes a framework for testing anomaly detection algorithms under identical conditions. Because quality test data representative of today's web servers is not available, this dissertation also describes the Hypertext Transfer Protocol (HTTP) request data collected from four web sites to use as training and test data representing normal HTTP requests. A collection of attacks against web servers and their applications did not exist either, so prior to testing it was necessary to also build a database of HTTP attacks, the largest publicly-available one.

These data were used to test nine algorithms. This testing was more rigorous than any performed previously, and it shows that the previously-proposed algorithms (character distribution, a linear combination of six measures, and a Markov Model) are not accurate enough for production use on many of the web servers in use today, and might explain the lack of their widespread adoption. Two newer algorithms (deterministic finite automaton induction and n-grams) show more promise.

This dissertation shows that accurate anomaly detection requires carefully controlled generalization. Too much or too little will result inaccurate results. Calculating the growth rate of the set that describes the anomaly detector's model of normal provides a means of comparing anomaly detection algorithms and predicting their accuracy. Identification of undergeneralization locations can be automated, leading to more rapid discovery of the heuristics needed to allow an anomaly detection system to achieve the required accuracy for production use.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Securing computer systems is important and difficult. The importance comes from both the potential losses of privacy or money from a compromised system as well as the civil and criminal liability for failing to secure computer systems specified in laws such as Sarbanes-Oxley [226] and the Health Insurance Portability and Accountability Act of 1996 [69]. The difficulty of securing existing systems is evident from the ever-expanding list of security vulnerabilities such as the Common Vulnerabilities and Exposures (CVE) [45] or the bugs discussed on forums such as Bugtraq [227]. As one measure of the rate of discovery of serious security problems, for the last several months Mitre has added 500–700 new candidate vulnerabilities each month to CVE [45].

Ideally, systems would be designed [184, 256], built [99, 184], and managed [36, 37, 105] in a way that prevents security vulnerabilities from existing. Unfortunately, the history of the last several years shows that production systems do not meet this standard. Clients, servers, and embedded systems have all had security vulnerabilities (a few examples are [29, 44, 55, 89, 229, 228]).

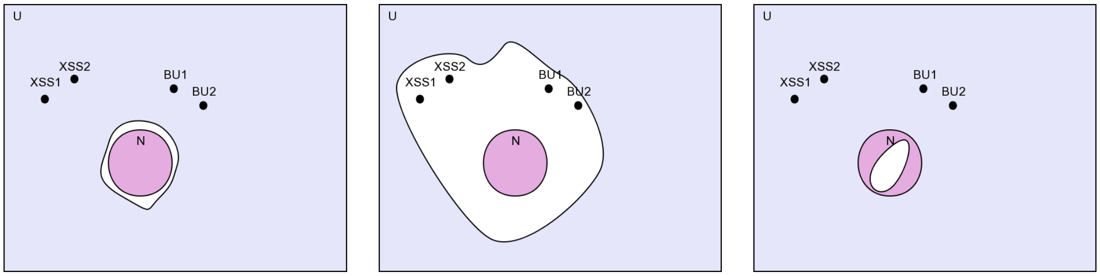
Since security is a requirement and we have yet to develop and deploy systems without serious security vulnerabilities, we need to take additional measures to protect the system against the next attack. One approach to protecting computer systems is to craft specific defenses for each observed problem, either in the form of code patches or attack signatures (e.g., [93, 199, 220, 241, 246]). Both strategies, however, require a human to analyze each problem and develop a solution. This limits the feasible response time to a timescale of hours or days, but attacks by self-replicating programs can spread in a matter of seconds [196]. Therefore, automated mechanisms that can detect and respond to threats in real time are needed. Anomaly detection systems have been proposed to fill this requirement because they can potentially detect novel attacks without human intervention [7, 91, 108, 177].

To better describe the proposed solution, Section 1.1 provides an overview of anomaly detection systems, with a focus on the concept of generalization and its importance for anomaly detection algorithm accuracy. Section 1.2, explains the motivation for my choice of web servers as the focused example for investigating anomaly detection and generalization. Because we have no general theory of intrusion detection, rigorous testing to compare anomaly detector models is a requirement. Section 1.3 further discusses the need for this testing and comparison. Section 1.4 summarizes the chapter and provides an outline of the rest of the dissertation.

1.1 Anomaly detection systems and generalization

An anomaly detection system develops a model of normal behavior from empirical observations (the **training set**); normal behavior implies that attackers are not using the system to perform tasks outside the set of those the administrators intend for it to perform. Subsequent observations that deviate from the model are labeled anomalies. An anomaly detection system developing a model of normal behavior is a form of machine learning.

If a learning system is to do more than simply memorize the training data, it must **generalize**, that is, generate a set that represents an example. When an anomaly detection system generalizes, it accepts input similar to, but not necessarily identical to, instances from the training data set, i.e., the set of instances considered normal (the **normal set**) is larger than the set of instances in the training data. For most anomaly detection systems (for example, those used with web servers), the set of possible legal input is infinite and the complete set of normal behaviors is not known and might be changing over time. In this case, the anomaly detection system must use an incomplete set of training data. For these systems, where the system accepts more instances than were provided in the training data, generalization is a requirement.



(a) (b) (c)

Figure 1.1: Diagrams illustrating (a) desired, (b) over-, and (c) undergeneralization. The points BU1, BU2, XSS1, and XSS2 represent attacks, N represents the set of normal requests, and the line surrounds the set accepted by the anomaly detection system. U is the set of all possible inputs to the system.

The goal for an anomaly detection system is a model that accurately describes normal behavior, as illustrated in Figure 1.1 (a). If the algorithm generalizes too much (**overgeneralizes**), then the normal set is too large. In this case, attacks “close” to the training data might be identified as normal (a false negative), limiting the usefulness of the system. Figure 1.1 (b) illustrates an overgeneralizing anomaly detection system.

On the other hand, a system that simply memorizes the training data will need sufficient storage for the complete set of normal. Obviously, this is impossible when the normal set is unknown or infinite. A system such as this would **undergeneralize**, and erroneously flag normal events as anomalous (false positives). An undergeneralizing system misses normal instances that are slight variants of training data. Figure 1.1 (c) illustrates undergeneralizing.

How easy it is to identify the correct level of generalization depends on the data stream and the representation used for it. Most data streams are not a collection of random bits; instead they consist of locations where values important to the meaning of the data are placed, i.e., they contain **structure**. An accurate anomaly detection algorithm will have a representation matching the data stream in a manner facilitating detecting anomalies. For example, consider a time and date stamp. In the data stream, it might be given as a number of seconds (or minutes, hours, days, etc) from a given starting time; for example, most Unix systems keep track of time as the number of seconds since midnight January 1, 1970. The time stamp might instead be presented in a more human-understandable format, such as HH:MM:SS MM/DD/YY. However, the order of these fields is different in different cultures, months and days might be by name or number, time might be indicated as B.C. or A.D. (with no year 0, so simple arithmetic on years fails without special processing). For example, in the data sent to web servers, three different time stamp formats are specified as acceptable by the standard, and even more are regularly used. Inside the intrusion detection system (IDS), the data might be represented as a string of characters, or a stream of tokens (e.g., hour, minute, second, day, month, year). The tokens might or might not also contain the associated value. Using tokens allows the algorithm to learn the structure, which brings the model closer to understanding the meaning of that which it models.

When considering generalization for a date, if it can only take on values from a finite set small enough to be held in memory, simply memorizing the string is sufficient, and no generalization is necessary. When any date is legal, the number of legal values is potentially infinite (although in practice it is bounded). In this case, tokens would be a better choice, and it might be the case that values associated with the tokens are not important for identifying normal behavior. The representation also relates to the expected anomalies. If a European date format is considered an anomaly (e.g., for a US system), then the representation must be capable of expressing this difference.

This example of a date illustrates the extremes between low (a small, finite set) and high (any legal date) variability of (a part of) a data stream. The problem is worse for data that is effectively random (such as cryptographic hashes or encrypted data). When an undergeneralizing anomaly detection system encounters high-variability data, it must attempt to memorize the data. In the worst case of random data (e.g., cryptographic hashes), this approach is doomed to use excessive memory and still fails to provide an accurate system. The more regions with high variability, the more combinations must be represented in the anomaly detector's model of normal behavior. The result will be a model that is too large to be practical, and/or one that produces too many false positives. If, on the other hand the anomaly detection algorithm generalizes enough to avoid the false positive problem, the less-variable portions of the data will likely suffer from overgeneralization and the system will be prone to missing attacks. High variability regions require more generalization than low variability ones. The problem, then, is how to identify the correct amount of generalization, and where to apply it.

In order to find the sweet spot, extra generalizations will be needed for an undergeneralizing system while one that overgeneralizes will need reductions in generalization. If the data have regions with different variability, these modifications must be localized to where the system over- or undergeneralizes. Additionally, to target these locations, the data representation must be fine-grained enough to distinguish between the portions of the data needing more and those needing less generalization.

To summarize, correct generalization is a necessary condition for accurate anomaly detection; for an anomaly detection system to have sufficient accuracy to be deployed, it must neither under- nor overgeneralize. If generalization is not properly controlled, then the anomaly detection will not be accurate. The model the anomaly detection system uses must represent the data in a manner allowing the discrimination of normal from anomalous data instances.

1.2 Web servers are a good testbed for anomaly detection research

My research uses web servers for investigating anomaly detection. They are a good test environment for several reasons. First, web servers are critical for many organizations. They are are vital conduits for communication, commerce, and services. Reflecting their high-value, attacks against web servers have become commonplace and expensive. For example, the Code Red worm cost 2.6 billion dollars in July and August 2001 [24], and Nimda[1] prevented hospital workers in Västra Götaland, Sweden from accessing reservations and computer-based medical records on September 23, 2001 [134]. While no lives were lost in this attack, it showed that costs could be more than simply monetary. Robertson et al. state that the CVE entries from 1999 through 2005 show that web-based vulnerabilities represent more than 25% of the total number of security flaws, and they note that this number is a minimum since it does not count vulnerabilities in custom web applications [218]. To attempt to defend against constant attacks[2], system administrators deploy defenses such as the *snort* network IDS [220]. *snort* uses patterns of attacks, called

[1] One method of Nimda's propagation was via attacks against Microsoft's web server, present on most machines running their operating system at the time.

[2] Yegneswaran et al. claimed in 2003 that there were on the order of 25 billion intrusion attempts per day on the Internet [274].

signatures, to identify attacks previously seen and analyzed. Each signature represents one attack; the signatures in recent sets show the fraction of attacks targeting web servers and their applications: 27% of the October 25, 2005 Sourcefire Vulnerability Research Team signature set [242] and 50% of the November 2, 2005 community signature set [43].

One reason for the large number of web application attack signatures is the variety of web applications, ranging from simple perl common-gateway interface (CGI) scripts to complex database-driven e-commerce sites. Because the tools for creating web applications are easy to use, many of the people writing and deploying them have little background in security (e.g., publishing, graphic design, or computer scientists with no secure programming background). Consequently, these web applications are often insecure [218]. The prevalence of custom web applications and their frequent security problems leads to the requirement for solutions such as IDSs to be applied as an additional layer of protection.

Adding to the problem is that HTTP has become the universal transport protocol. Previously, for each new service, a software developer developed a communications protocol and used an assigned port for communications. Because of security issues associated with many new protocols, network administrators blocked access with firewalls [121, 122]. HTTP travels through most firewalls, often with little interference. This led to developers using HTTP as a transport protocol for their new software. Examples include SOAP[3] (remote procedure calls using the extensible markup language (XML) for parameters) [10], tunneling secure shell (SSH) connections [265], Apple's QuickTime multimedia [11], Microsoft RPC for accessing Exchange (email) servers [192], a protocol for negotiating payments [78], and remote filesystem access [141]. These new services over HTTP present additional opportunities for attackers.

Current IDS approaches for web servers are insufficient. An web server IDS should:

1. be accurate—identify most, ideally all, attacks, and rarely misidentify non-attack traffic. It must maintain this accuracy as the web site changes over time.

2. detect novel attacks.

[3] In 2003, this acronym was declared to have no meaning by the Worldwide Web Consortium (W3C).

3. not place an undue burden on the administrator or the machine it is protecting.

As Chapter 2 and experimental results in this dissertation show, current IDS approaches for web servers do not meet these criteria. One possible reason is that, as Section 2.4 shows, HTTP is a difficult protocol for anomaly detection.

1.3 IDS testing

Other researchers have proposed anomaly detection for HTTP IDSs (see Chapter 2 for details). Unfortunately, comparisons of of IDSs using the same experimental data are rare. In some cases, the researcher(s) used a data set not available to other researchers (a **closed** data set) and showed that the IDS can detect attacks. In other cases, the researcher(s) used data sets available to all researchers (**open** data sets) such as the MIT Lincoln Labs data (Section 2.3.2 describes this data set and its shortcomings). Papers describing these IDSs only report results on the data set, and comparisons are usually left to the reader to perform. Therefore, how can we really know how accurate the IDSs are?

A theoretical basis for intrusion detection would make testing less important, but, as I will explain in Chapter 2, we have no such theory. Therefore, I used an experimental approach. As a part of the research presented in this dissertation, I used a framework to test nine algorithms to evaluate how they performed under realistic circumstances. This framework, the IDS algorithms, and the HTTP parser are open-source to encourage other researchers to expand upon my testing.

One of the "best practices" for system administrators is that they must regularly apply patches to protect their servers (e.g., [193]). Chapter 4 shows that Internet-connected web servers are regularly attacked with old attacks and attacks for other web servers—attacks to which a properly patched server is immune. When an anomaly detection system is used with an Internet-connected server, it should not produce alarms on harmless attacks. This requirement further challenges the developer of an anomaly detection system. Previous IDS approaches have used training data where these old attacks have been filtered out. Requiring attacks to be filtered from the training data means that first, a source of attack-free data must be found—accurately removing attacks while retaining the complete HTTP request is difficult and is likely to require hand-tuning of a signature database. Secondly, an IDS trained on this data will produce alarms for the harmless attacks present in normal web requests. Also, most administrators do not want to be alerted to every harmless attack[4] (see Section 4.3 for examples of the rate of harmless attacks). For an IDS to be deployable, these attacks should be allowed in the training data and the IDS algorithm should be able to still recognize novel attacks—in other words, the learning algorithm must be able to tolerate old attacks as normal yet still be able to detect new attacks.

1.4 Summary and dissertation overview

Web servers are important to protect because they are ubiquitous and critical for many organizations, yet they are currently poorly defended. Web servers are a good testbed for anomaly detection research because of the impact a good solution can have on production systems. Most deployed defenses are signature-based, and these systems require humans to analyze attacks and they cannot detect novel attacks.

Anomaly detection techniques can detect novel attacks, and hence are a promising solution, and one a few researchers have tried. Anomaly detection systems make use of generalization whenever the normal behavior is infinite, unknown, or nonstationary. Generalization must be carefully controlled for acceptable accuracy.

To date, IDS testing has been weak, yet rigorous testing is required to know how accurate the system is and under what circumstances it works well. Testing should include harmless attacks, since they are a normal part of Internet traffic.

The rest of the dissertation is organized as follows. Chapter 2 sets the stage for my research by reviewing prior research; it also introduces HTTP and shows why HTTP is a difficult protocol for anomaly detection systems. Chapter 3 described the algorithms I tested, while Chapter 4 describes the test framework and data sets. Chapter 5 describes the tests I performed and presents the results of these tests. Chapter 6 discusses the meaning of the test results, and shows how to compare anomaly detection algorithms by comparing

[4] Administrators wising to see old attacks can make use of a signature-based IDS, which is well-suited for this task.

how they generalize. Chapter 7 concludes the dissertation by summarizing my contributions and looks at additional avenues of research highlighted by my work.

Chapter 2

Background

As one part of protecting web servers, a system administrator needs to know when her system is under attack and has been (or is in danger of being) compromised—this is intrusion detection. Chapter 1 introduced this problem; here we consider the current state of the art. This chapter surveys the literature describing previous research results, starting with intrusion detection systems in Section 2.1.

Chapter 1 argued that an IDS protecting web servers must detect novel attacks without human intervention, and that anomaly detection systems are a solution that addresses this requirement. All anomaly detection systems share a common trait of learning a model of normal behavior. Over the years, researchers have tried many different algorithms for learning this model. Some of these algorithms have promise for HTTP; others have limitations that prevent them from ever working in this domain. A summary of these approaches appears in Section 2.2.

Whenever the set of training data representing normal behavior for the anomaly detection system is incomplete or the actual set is infinite, the anomaly detection system must perform generalization. Although researchers have realized the usefulness of generalization, the extent to which they have investigated it is limited. Section 2.2.2 reviews the previous work in this area.

The large variety of anomaly detection algorithms is a symptom of the fact that we lack a comprehensive theory of intrusion detection and anomaly detection to provide guidance on approaches with more promise. Section 2.1.3 shows that most theory has been developed in Ph.D. dissertations, and none comes close to being comprehensive.

Lacking theory, rigorous testing and comparison of algorithms is necessary to determine which systems work well, when, and why. Section 2.3 reviews the prior work and concludes that the quality of testing and test data vary, and repeatable, comparable results for HTTP are rare.

Section 2.4 provides background information on the HTTP protocol, structure in the HTTP requests that might be useful for an anomaly detection system, sources of diversity between web sites that an IDS might use to force an attacker to customize an attack for a given web site, and examples of attacks against web servers. The section ends with information about the difficulties HTTP poses for IDSs.

2.1 Intrusion detection

Computer intrusion detection started in 1972 with a paper by Anderson [7] identifying the need for what would evolve into today's intrusion detection systems. Early IDS researchers focused on statistics of system and user behavior on a given machine (a host-based IDS) to address insider threats [68, 126, 177, 200]. In practice, these early systems had high false-positive rates [16, 27, 30, 39, 47].

As networks connected machines together, network servers became common and attackers could be physically removed from their victims. This led to the need for network-based intrusion detection, and researchers tried attack detection and prevention at many of the layers in the ISO model of networking [123]. Physical access restrictions (traditional locks and other physical security) function well for preventing attacks at the physical layer[1]. Approaches included identifying unusual traffic based upon its source and destination IP addresses, protocol (TCP or UDP), and ports [19, 108, 115]. Unfortunately, nothing is unusual or particularly dangerous about a new machine connecting to a web server, so another approach is needed for protecting web servers.

[1] Physical protections work well, assuming the physical layer is copper or fiber optic; wireless networks are attacked at the physical layer, but these attacks are beyond the scope of this dissertation.

The inability to protect web servers led to explorations of other approaches. Some researchers restricted HTTP to a presumably safe subset of the protocol [80]. Other approaches monitored the HTTP data stream at the ISO network application layer [123], in spite of the perceived difficulty of using this data stream [276]. Some of these approaches treated web servers as a generic network service. Incoming and outgoing traffic were modeled as a stream of bytes or as discrete packets. Some approaches looked for patterns in the first few packets of connections [178]. Others compared character distributions between the payloads of similar-sized packets [263].

In contrast with these protocol-independent approaches, some researchers focused on the HTTP requests in the network application layer, for example combining statistical characteristics of common gateway interface (CGI) program request parameters [146, 147]. However, these anomaly detectors must be trained on data without attacks. This requirement is problematic because the normal background of today's Internet contains large numbers of old attacks, most of which are ineffective against properly patched servers. Signature-scanning intrusion-detection systems (IDS) such as *snort* [220] can be used to filter out known harmless attacks; however, the high accuracy required for training requires frequent updates to the attack signature database and careful site-specific tuning to remove rules that generate false alarms. This manual intervention reduces the main advantage of using anomaly detection.

2.1.1 Review papers

IDSs have existed since the 1970s, and therefore many papers have been written about them. Several people have reviewed the state of the art, including: Allen [2], Bai and Kobayashi [18], Bilar [26], Jones [130], Kemmerer [135], Kvarnström [133], Lunt [176], Mukherjee et al. [197], Singh and Lakhotia [235], Turkia [250], and Verwoerd and Hunt [255]. The best reviews are those that present an unbiased, thorough review of the literature, and/or provide a good taxonomy for describing different intrusion detection methods. Examples of such reviews include those by Axelsson [13, 17], Debar [63, 64, 65], Alessandri [1] (who works or worked with Debar), and McHugh [187]. Although not a review of research IDSs, Jackson [124] wrote an excellent in-depth survey of commercial products. Brumley et al. reviewed defenses against worms, many of which are variants of intrusion detection systems [33].

2.1.2 Architectures

As researchers have worked on the problem of intrusion detection, they have developed three IDS architectures [13, 17, 18, 63, 64, 65]: signature detection, specification, and anomaly detection. A few researchers have tried hybrid approaches, where the strengths of one architecture is exploited to cover the weaknesses of another.

Signature detection (also called misuse detection, or detection by appearance) A human studies an attack and identifies the characteristics (e.g., behavior and/or content) that distinguish it from normal data or traffic. The combination of these characteristics is known as the *signature*, and it becomes part of a database of attack signatures. When the IDS encounters data matching the signature, it raises an alarm. Signature systems represent the vast majority of installed IDSs; they are important, though limited, as the next paragraphs show. All commercial anti-virus products make use of signature detection, as does the network IDS *snort* [93, 220].

Signature-based systems are usually fast, and they can often detect accurately the attacks for which they have signatures. However, because a human must analyze each attack to develop the signature, the feasible response time to new attacks is limited to a timescale of hours or days. Attacks by self-replicating programs (viruses or worms) can appear and spread in seconds [196]. When new attacks appear, systems protected by signature IDSs are vulnerable until an updated signature set is available and installed; i.e., they fail to meet criteria 2 in Section 1.2.

All signature-based IDSs use some form of pattern matching to identify the strings that might be associated with an intrusion, be it a virus in a file or an attack such as Code Red against a web server. In many pattern-matching systems, a pattern is simply a string (ideally) not found in any normal data. Some pattern-matching systems include other information, such as where to look for the string (either by absolute location, or by location relative to the network protocol or file format). Pattern matching approaches include colored petri nets [46, 148, 149, 150, 151] and attack specification languages [73, 79, 232, 257, 258, 259].

Signature-based systems suffer from false positives, especially if the system configuration or environment changes. Patton et al. described this phenomenon that they call **squealing**, and showed how intruders can negate the benefit of a signature-based IDS with carefully crafted false positives [207]. IDSs with this problem fail to meet criteria 3 in Section 1.2.

Signature-based systems can also suffer from false negatives. When testing to see the effectiveness of the two "best" signature-based systems (*snort* and ISS RealSecure), Vigna et al. found that mutants of known attacks would not be detected by the IDS [260]. IDSs with this problem fail to meet criteria 1 in Section 1.2.

Most signature systems do little, if any, generalization. One that does is reported by Ning et al. [204]; they abstracted signatures to help detect related attacks. An interesting test, one apparently never performed, would be to apply the work of Vigna et al. [260] generating mutants of attacks to the Ning et al. work where the signatures are generalized.

Specification An expert studies the program to be protected and produces a specification that identifies the allowed (i.e., those intended by the program author) actions that a program can take. When the system is running, the behavior is compared to the specification, and deviations are noted and/or prevented [142, 231, 233, 234]. The complement of this approach is describing behavior indicative of attacks; sometimes describing what is not allowed is easier than describing what is not allowed [209, 210].

A problem with this approach is that the expertise level necessary to develop a proper specification is considerably higher than that needed to implement a program. Additionally, each program needs a separate specification, and any time the program changes, the specification must be reviewed by an expert. These requirements raise the cost, and hence have lowered its use in production systems.

Specification-based systems also face problems with their proper use. For example, Hogland and McGraw note that access control lists, one of the simpler specification-based security measures, are so complicated that they normally fail in practice [117].

Assuming a proper specification, such an IDS meets the criteria from Section 1.2.

Anomaly detection (also called detection by behavior) Anomaly-based IDSs assume that intrusion attempts are rare and that they have different characteristics from normal behavior. The IDS induces a model of normal from a corpus of training data. When an instance that does not match the model learned from the training data appears, the system raises an alarm. Finding the proper representation for the data and identifying a learning algorithm to use with the data can be difficult for anomaly detection systems. A well-designed anomaly detection system can meet all of the criteria specified in Section 1.2. They are one of the major foci of this dissertation, and hence are covered in more detail in Section 2.2.

Hybrid systems An IDS using this architecture uses parts from two or more of these architectures to take advantage of the strengths of one architecture to cover the weakness of another. As one example, Sekar et al. combined statistics for anomaly detection with a human-designed state machine specifying the allowed network communication [233].

Only one hybrid system has been reported for HTTP. Tombini et al. [248] combined misuse and anomaly detection to find attacks in logged HTTP requests, and they analyzed how to resolve conflicts between the two architectures to provide the best accuracy. This system suffered from several drawbacks that would hinder its deployment in a production environment. First, they reported that out of 56 attacks, the misuse detector could detect 30—just over half. They did not publish results that tested how the combined system performed on attacks. In addition, they deleted dynamic user account data from their test suite (data I retained in my test set), using only the dynamic content on the official web site. They also manually adjusted the model induced by they anomaly detector. They also used manual methods to identify safe or dangerous web requests, reporting that a person can perform this task within a few days. This heavy reliance on humans limits the usefulness of their system.

Although this dissertation does not study hybrid IDSs, if an anomaly detection or specification system is one of the architectures in the hybrid, the resulting IDS can meet all of the criteria specified in Section 1.2.

2.1.3 Theory

Theory for intrusion detection is important because it could predict when various algorithms will work well, when (or if) an IDS problem is solvable, and when we must be content with a “good enough” but less than perfect solution. Few researchers have worked on a theory of intrusion detection, and none has generated a broad theory that would answer these questions.

Lee and Xiang [166] studied the entropy of IDSs. This information-theoretic approach predicts whether an algorithm will be suited for use with a specific set of data. It also predicts a window size to use when looking at the data.

Brumley et al. [33] developed a theory to determine the approaches to defenses against worms most likely to work. Since some defenses against worms are IDSs, their work applies to a portion of existing IDS approaches.

Forrest et al. [92] proposed an immunological approach to computer security using negative detection—in the example presented in the paper, they generated a set of strings not matching strings in files on a system. If these strings showed up in a later scan, an anomaly was noted. Their analysis included a formal look at the relationships between:

- the probability of detection
- the number of detectors needed
- the detector length.

This work was followed by D’haeseleer et al. [71, 70]. They studied the information theoretic implications of this work and explored the existence of **holes**—non-detectable strings in the non-self set. Esponda et al. [85] continued this work by:

- giving a formal framework for analyzing the tradeoffs between positive and negative detection schemes in terms of the number of detectors required to provide a given accuracy of detection,
- introducing a new matching rule, r-chunks,

- showing that permuting the representation of the pattern strings reduces the number of holes, and
- showed how many permutations are required to minimize the holes.

Many people working on graduate degrees in computer science have incorporated theory in their theses or dissertations. Some examples include:

- Hofmeyr [113] analyzed of the negative detection system and its the data representation that he developed. The analysis showed the number of detectors required for coverage of the space to be as complete as possible. His analysis also looked at the holes and the tradeoffs associated with closing these holes. He also analyzed the convergence properties of his data using a stochastic model; this analysis is also useful for understanding the nonstationary nature of his data.
- Lane [157] analyzed the nonstationary nature of the data existing in many intrusion detection data sets, and in particular, user command-line usage. The data changes over time as users learn new ways of solving problems, software on computers is updated, etc. The result is that normal behavior changes and the data source is therefore nonstationary.
- Lee [161] provided a formal analysis of data-mining techniques for intrusion detection. Without such a discussion, one would wonder how well such an approach can do. His theory supports the potential for data mining to detect abnormal behavior.
- Somayaji [239] formally compared two methods of IDS data modeling of system calls, one using sequences calls and one using lookahead pairs (the call at the current position and the call at position k steps ahead of the current call), showing that the two models were equivalent.

Nobody has developed a general theory of intrusion detection. Nobody has formally proven whether intrusion detection or anomaly detection is a solvable problem. The theoretical side of intrusion detection still needs the most basic work.

2.2 Anomaly detection

Because anomaly detection holds the potential for detecting novel attacks, many models and algorithms have been tried. An anomaly detection system uses a body of training data to induce a model of normal, and then it flags as abnormal any data instances not included in this model. Details about how anomaly detection works were presented in Section 1.1.

The first anomaly detection systems were based on statistical measures of normal (e.g., [4, 5, 6, 68, 126, 177, 267]) with host audit records and lower-level network data as the data source. Forrest et al. [91] introduced the idea of modeling program behavior by recording short sequences of system calls. This work generated significant interest and many proposals for alternative models of system call behavior, including variable length sequences [181] and rule-based data mining [164]. Unfortunately, system call monitoring does not appear to be a promising approach to detecting attacks against web servers. Earlier work with pH [239], a Linux-based IDS that monitors sequences of system calls, suggests that large server programs such as web servers are difficult to monitor in practice, sometimes taking months to achieve a stable model of normal behavior (although see [230] for one idea about how to address this problem).

Another problem is that many attacks against web servers are caused by configuration mistakes or errors in the web application code. Exploits based on these vulnerabilities might not generate unusual system call patterns because they do not cause unusual execution of web server code. The web server is simply interpreting the vulnerable script, and the system call trace generated by the interpretation process is similar whether or not the script contains a vulnerability.

Because all I/O behavior typically passes through the system call interface, virtually all attacks on web servers should, in principle, be detectable by observing arguments to system calls. Earlier attempts to model system call arguments, however, generated high false positives [145], despite the use of a much more sophisticated modeling strategy than Forrest et al. originally proposed. This discrepancy appears to be a natural consequence of the high variability of system call arguments relative to sequences of system calls.

2.2.1 Algorithms

Over the years, researchers have proposed many algorithms for anomaly detection; some of these algorithms have limitations (e.g., requiring fixed-length input when HTTP is variable-length) that prevent their working with HTTP. Because results showing an algorithm that fails in a given situation are difficult to publish, we see only when an algorithm can be shown to perform adequately on a given data source and representation; i.e., it is accurate at least for the tests performed. In the algorithm descriptions that follow, I focus on those that have been tried with HTTP, and I comment on the algorithm's suitability for use with HTTP

Rule-based systems

Rule-based systems consist of a set of rules describing normal behavior. Given a set of observations of normal behavior, the rules describe the data. For example, if the data include session start and end times, the IDS might generate a rule specifying that sessions do not start before 6:00am or that they do not last more than eight hours. The rules might be relate multiple, distinct events, for example, operations x, y, and z only occur within sessions of type w. Once the rules have been generated, the IDS generates an alarm whenever a rule is broken.

One type of rule-based system is an expert system. In this case, the rules are generated by humans [4, 5, 6, 61, 67, 68, 111, 118, 120, 125, 169, 170, 177, 200, 212, 213, 237, 238, 249, 253]. The alternate approach is to have the IDS automatically induce the rules [110, 161, 162, 163, 165, 247, 251, 272].

A thorough search of the literature revealed only one HTTP IDS using this approach. Vargiya and Chan [254] compared four statistical techniques to determine token boundaries automatically: boundary entropy, frequency, augmented expected mutual information, and minimum description length (MDL), as well as two combinations of these methods. They determined that frequency and MDL performed best at automatically identifying tokens. The tokens were used with a rule-generation system, LERAD [180].

Descriptive statistics

In a statistical system, the IDS observes a set of normal behavior and calculates one or more statistics identified by a person or some other portion of the IDS to be significant. The statistical methods used might be simple frequency analysis, Bayesian belief networks, Hidden Markov models, or other measures. The IDS then notes deviations from these statistics. Many earlier approaches were statistical [4, 5, 31, 32, 34, 59, 61, 72, 75, 76, 82, 83, 84, 86, 90, 106, 107, 109, 118, 119, 126, 127, 143, 157, 175, 177, 188, 189, 190, 191, 200, 224, 225, 236, 252, 267, 268, 269, 270, 271, 272, 273, 275, 277].

When applied to HTTP, most researchers have analyzed the data stream as a sequence of characters, and applied statistics to the characters. For example, Wang and Stolfo [263] modeled the character distribution of different sized packets. Mahoney and Chan [179] performed statistics on packet headers, with some statistics on the application-layer data (e.g., HTTP keywords). Mahoney [178] then modeled the first 48 bytes of the application layer data of interesting services (including HTTP) using a collection of statistical measures.

Kruegel et al. [146, 147] analyzed common gateway interface (CGI) program parameters using a linear combination of six measures: attribute length, character distribution, using a Markov model, labeling whether tokens were random or from an enumerated set, attribute presence or absence, and the attribute order.

Estévez-Tapiador et al. also used a Markov model [86]. They tried the model with the characters from the request, and found it performed poorly. However, after breaking the request into pseudo-tokens, performance improved. Unfortunately, their best reported false positive rate was 5.76%; for the cs.unm.edu web server (Section 4.3), this would correspond to a false positive rate of over 5,000 alarms per day.

Directed graphs

Some representations explicitly encode the successor relationship as a directed graph. For example, three approaches used in HTTP intrusion detection use this representation: the Markov model discussed previously, Deterministic Finite Automata (DFA), and n-grams. Both the Markov model and the DFA consist of a set of nodes, with transitions between the nodes. An n-gram is a sequence of n items, characters or tokens; a set of n-grams is the model for normal. In n-grams, item a following item b is encoded by ab as a part of one or more of the sequences in the set.

Wagner and Dean used a directed graph to represent the system calls made by a protected program [261]. Kruegel and Vigna and Kruegel et al. [146, 147] used a digraph to represent the order of CGI parameters in the HTTP requested resource path (implementation details are in Section 3.3).

Deterministic Finite Automata (DFA) For this approach, the IDS induces a DFA from its training data, effectively learning a (formal) language. Strings that are not a part of the language are labeled as anomalies. When the data being modeled are described by a language (e.g., network protocols), this approach learns the subset of the actual protocol used by the computer(s) that the IDS is protecting [25, 181, 190, 191, 230, 234]. Other researchers (e.g., [144, 230, 181]) have studied anomaly detection using methods for constructing finite automata-based models of program behavior.

In the worst case, learning a DFA from only positive evidence (one-class learning) is known to be NP-complete [98], but Lang showed that the average case is tractable [158]. In practice, DFA induction is practical in many contexts, and there is an extensive literature on DFA induction algorithms (good overviews include [38, 159, 206]).

n-grams and lookahead pairs n-grams have been used in information retrieval [183, 194], measuring document similarity [57, 87] or categorizing text [42] and similarly for a measure of normal in intrusion detection [91, 92, 114, 116, 128, 129, 181, 182, 190, 191, 230, 239]. Lookahead pairs are similar in effect to n-grams, but Somayaji [239] found that they were more efficient.

Neural networks

One or more data sources is used to train the neural net to recognize normal behavior. The neural net then identifies behavior not matching its training experience [35, 62, 74, 82, 94, 95, 160, 198, 222]. In other fields, neural networks have shown their ability to detect patterns. Most neural nets require fixed-length input [195], a requirement that could cause problems with variable-length data such as HTTP requests.

Support Vector Machines

Noting that some anomaly detection algorithms undergeneralize, Mukkamala et al. [198] used a Support Vector Machine (SVM) to learn the training data from the 1998 DARPA/MIT Lincoln Labs IDS test [171]. These data included two HTTP attacks. They compared the SVM with a neural network and found that the SVM was more accurate and faster. The SVM training time was 17.77 seconds compared with the neural net training time of 18 minutes.

Information retrieval

In addition to n-grams, other techniques from information retrieval (IR) for categorizing text and measuring the distance between different texts might be useful for categorizing data. In spite of this parallel, only a few researchers have tried it, none with HTTP [8, 152, 153, 155, 154, 156, 157].

Multiple sensor fusion

Axelsson [14, 15, 16] shows that to avoid the base rate fallacy[2] and hence generate too many false positives, an IDS must have high accuracy (or low error rate). To address this problem, one approach is to combine results from different intrusion sensors [21, 22, 23, 47, 96, 146, 147, 169, 170, 200, 202, 212, 213, 238, 253].

[2] The base rate fallacy is when a detector has what appears to be good accuracy, but the rate of what it detects occurs infrequently resulting in poorer performance than might be expected. For example, on 1,000,000 samples, a detector that is 99% accurate for something that occurs once in 10,000, the detector produces a true positive only 1% of the time. See Axelsson's papers for more details.

Immune system

The immune system might be viewed as having achieved the goal that IDS researchers are trying to attain. The ability to recognize and destroy hostile entities (non-self) while not attacking the organism (self) is the holy grail of intrusion detection [60, 92, 104, 112, 113, 114, 136, 137, 138, 139, 140, 208, 217, 239, 240, 266]

An intriguing mechanism of the immune system is negative detection. In the vertebrate immune system, lymphocytes have detectors for molecules or peptide fragments[3] that are not associated with the individual (i.e., foreign). Taking a cue from this portion of biology, some systems use negative detection to generate detectors for data that do not occur in the normal data stream. When one of the detectors matches, an anomaly is presumed to exist. Negative detection has the advantage that it can be easily distributed across all the machines on a network. Forrest et al. [92] used negative detection for virus detection, and Hofmeyr [113] used it for network traffic pattern anomaly detection. Unfortunately, this approach does not work for web servers, where novel connections are normal.

Another immunologically inspired idea is that of tolerization. When the IDS generates a detector, it is considered naive. If a naive detector matches a string early in its lifetime, the detector is considered to be matching self, and it is deleted. If a detector survives the tolerization period, it becomes mature. When a mature detector matches a string, the request is considered anomalous.

In the immune system, co-stimulation of T- and B-cells is required from other cells before an immune response is generated. In LISYS [112, 113, 114], a mature detector that matches a request is deleted unless it receives co-stimulation. This idea is related to that of sensor fusion, although in this case, the second sensor is a human.

2.2.2 Generalization

Overall, many researchers have mentioned generalization, but few attempted to control it, understand it, or detail its relationship with accuracy.

[3] Peptide fragments are pieces of proteins generated by a cell in its normal operations. Proteins are broken into pieces and displayed on the outside of the cell by class I major histocompatibility complex (MHC) molecules.

Most earlier anomaly detections systems assume that the data they use is stationary. Researchers working with nonstationary data use generalization in order to tolerate the novel instances that are a hallmark of nonstationary data. Some researchers working with nonstationary data include Mahoney and Chan [179], who used an exponential decay of learned probabilities of features in network data. Lane (sometimes with Brodley) showed that user event data is nonstationary, and they identified methods for measuring the magnitude and direction of the drift [152, 153, 155, 154, 156, 157]. Much of the work on nonstationary data focuses on eliminating the old portions of the model (forgetting), e.g., work by Salganicoff [223]. Littman and Ackley [174] looked at cases in which the problem can be divided into two parts, variable and invariant, although this approach would not apply to environments (such as HTTP) where little is invariant. Denning [68] described a system that modeled the data recorded by an audit system. The data were generalized by several statistical measures of patterns in the audit records. She recognized the problem presented by nonstationary data (e.g., adding new users to a protected system), and she proposed approaches that might improve the situation. However, she did not report results showing the effectiveness of these approaches. She, like most of the following researchers, did not characterize how the generalization chosen affected accuracy of result.

False positives are an indication of undergeneralization or insufficient training. Axelsson [14, 15, 16] noted that an intrusion detection system must be very accurate to avoid producing many more false alarms than true positives. In addition to controlling false positives through targeted generalization, some anomaly detection systems use techniques to control false positives. Two approaches have been proposed to address this problem. The first is sensor fusion, as described in Section 2.2.1.

The second approach is correlating alarms. The idea is to group alarms into classes representing behavior, and assume that all of the alarms in one class represent the same behavior. The idea is that a system administrator can view a single exemplar from the class and determine if it represents normal or abnormal behavior [47, 48, 56, 113, 131, 132, 167, 202, 203, 243, 253]. Robertson et al. [218] went one step further, using heuristics to identify the attack type associated with a class of HTTP requests. Julisch [131] looked at the problem of false alarms overwhelming human operators. He clustered alarms to identify the root cause and was successful, reducing the future alarm load by 82%. The theoretical aspect of his paper showed that general alarm clustering is $\mathcal{NP}$-complete.

Li et al. [168] are some of the only researchers to investigate the relationship between generalization and anomaly detector accuracy. Noting that generalization can (at times) improve accuracy, and they showed that generalization on normal data is good, while generalizing on attack data is not (this result assumes a suitable set of attacks exists). This result contradicts the work reported by Anchor et al. [3], who discussed generalizing detectors for attacks, with the goal of improving an anomaly detector at detecting new, similar attacks in network packet data. Robertson et al. [218] also generalized anomalies with the goal of generating "anomaly signatures" to find further attacks of a similar nature. None of these researchers characterized the amount of generalization (e.g., through the size of the set described by the model), nor did they consider controlling generalization to improve accuracy.

2.3 Testing intrusion detection systems

There are at least two reasons to testing IDSs:

- to verify that an algorithm is able to detect attacks.
- to compare two or more algorithms to determine the best one under various metrics.

Most researchers test algorithms to support a claim about a particular algorithm. This testing is frequently little more than asking, "Does the IDS detect an attack?" Slightly better are researchers who ask the question, "Which of the following attacks can the IDS detect?" Even this testing is often acknowledged as weak, as shown in the following quotes:[4]

> Initial testing shows the algorithm performs satisfactorily.
>
> Preliminary experiments prove...
>
> The preliminary experiment results show the effectiveness of our system.
>
> A realistic UNIX system is much more complex than the one that we have modeled in this paper.

[4] References withheld to protect the guilty.

When studies compare two or more IDSs, they are sometimes only to justify why the author(s) approach is the "best" one, and not a real comparison at all. In other cases, the authors present a "straw man" argument, in which the algorithm chosen for comparison is unlikely to work well in any production environment. These types of tests show only that their IDS is better than an incompetent algorithm.

One place where comparisons of multiple IDSs under identical circumstances can be found is in the trade literature, where commercial IDSs are compared with the intent of finding the "best" product. These types of comparisons are not limited to the trade literature, as sometimes they appear in peer-reviewed journals or conferences. Examples of commercial IDS comparisons include Newman et al. [201] and Rae and Ludlow [216].

As we can see, careful IDS testing is rare. In their review of IDS testing, Athanasiades et al. state that they do not believe this problem will ever be properly solved [12]. There are several possible explanations for the scarcity of good IDS testing:

- According to Debar, a set of criteria for evaluating an IDS does not exist [66].
- Identifying appropriate data is difficult—the data must be representative of realistic operating conditions. Data collected live from a network might be subject to 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 a good, protected network is resource-intensive, both in the costs of the hardware, as well as human time to set up and maintain a diversity of machines needed to ensure a good test environment.

  Maintaining a collection of vulnerable machines is also difficult. Bugs are discovered as systems evolve. The result is that exploits are often specific to a given operating system (OS) distribution and version, as well as the versions of compilers, libraries and other software installed. Since a given machine (or virtual machine) can only run one version of the operating system at a time, researchers need substantial resources to maintain a collection of vulnerable machines.

- Many intrusions are fragile; if any part of the environment for which the intrusion was written is not as expected, the intrusion is likely to fail. Either there is no damage (e.g., only a log entry indicating something odd occurred), or the intrusion attempt turns into a denial-of-service attack. This fragility means that maintaining a good collection of attacks requires work to ensure that they function in the test environment.

These difficulties should not be considered excuses, as some researchers *have* done a good job at comparing IDSs (e.g., [264]). Good testing is repeatable; the data are available to other researchers so they can directly compare the results of new ideas, 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.

2.3.1 Frameworks for testing

In order to test multiple IDSs, one of the requirements is that the data and environment be reproducible. A framework for testing is one way of achieving this reproducibility by providing a setup in which different IDSs can be tested under identical conditions. Four researchers or research groups have established such frameworks:

- The first published papers about an IDS testing framework and methodology were from Puketza et al. [214, 215] at UC Davis. Unless they failed to publish further work, they built the framework and then tested only one IDS: NSM [106, 107].

- Wan and Yang [262] 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) [49]. Their framework might be useful, but the paper describes only a preliminary version.

- IBM Zurich [66] 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 is a source of what could be normal.

2.3.2 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 that are in common use today. Given the diversity between web sites, the best state of affairs is to use data collected from the server to be protected. These data often have privacy issues associated with it, 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 accurate data sets.

The most famous open IDS data sets are the DARPA/MIT Lincoln Laboratories IDS tests of 1998 and 1999 [77, 101, 102, 172, 173]. The researchers who generated these sets collected data from computer networks, where some of the machines are attackers, some victims, and some exist simply for traffic generation to make the detection task more difficult. These data have since been used by many other researchers as a test dataset for IDS ideas, because it is easy to see how the IDS would have fared, had it been in the original Lincoln Laboratories test. The ability to reuse data allows comparison between methods, but in order to make the comparison, a researcher must collect all of the relevant papers and then combine the results to generate the comparison.

This data set is not without its critics. McHugh [185, 186] 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.

This data set is also quite dated, as web behavior has evolved significantly over the years. In contrast with web servers in 1999, custom web applications are commonplace today, and attacks target these applications as well as the server itself. Finally, the union of both the 1998 and 1999 data sets contains only four attacks against web servers. At the time the data was collected, web servers represented a smaller portion of an organization's network servers. When systems developed using these data are tested on a broader data set, their performance suffers; confirmation of this assertion appears later this dissertation. In spite of these limitations, Wang and Stolfo [263], Mahoney [178], and Mahoney and Chan [179] Vargiya and Chan [254] 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. [86] 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, for privacy reasons, these data are unavailable for others to use, leading to the inability to compare IDS approaches directly. Other researchers gathered their own data. Kruegel et al. [146, 147] tested their system using extensive normal data sets from multiple sites (including Google). Unfortunately, these data are not available for other researchers[5]. Another good part of their test data is that for a portion of their 12 attacks, they used attacks against software that ran on one of their data source web servers. Tombini et al. [248] 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. Again, for privacy reasons, these data are not available to other researchers. Estévez-Tapiador et al. [86] used 1500 attack requests representing variants of 85 distinct attacks, giving them 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 [54] exhausts Apache's memory by a repeated header line. Wang and Stolfo [263], in different experiments, modeled the packet payload, the first and last 100 bytes, and also the first 1000 bytes of the connection. Kruegel and Vigna and Kruegel et al. [146, 147] obtained their test data from web server log files, and only looked at CGI programs. Web server log files are a popular data source, and Tombini et al. [248] and Robertson et al. [218] 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. Another drawback with this data source is that by the time an attack appears in the log file, it is too late to stop it. However, an approach

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

validated on log files could be reimplemented in front of the web server to protect it.

Testing with harmless attacks in the training data

One key problem not addressed by earlier anomaly detection systems for HTTP is the presence of benign attacks in normal training data (introduced in Section 1.3). All of the earlier IDSs pre-filtered attacks from the training data before building models. The researchers using the MIT Lincoln Labs data used attack-free training data [263, 178, 179, 254]. Other researchers used web server log files, that allowed them to filter out invalid requests (some fraction of which are the harmless attacks) [146, 147, 218, 248].

2.4 HTTP

My research uses HTTP to investigate anomaly detection algorithms. The information in Section 2.4.1 describes and provides examples of the protocol. The protocol contains structure; examples of the structure and illustrations of the different levels of variability in the structure illustrate this feature of the data in Section 2.4.2. Section 2.4.3 introduces the sources of diversity between web sites; an IDS using this diversity might be able to force an attacker to customize an attack for each web site. Many attacks against HTTP exist; Section 2.4.4 shows several and illustrates the diversity between attacks. Several features of HTTP make it a difficult protocol for an IDS to use successfully; some of these features eliminate algorithms from consideration. Section 2.4.5 explores this further.

2.4.1 The HTTP protocol

HTTP is a stateless protocol described by the Internet standard RFC 2616 [88] with extensions described by other standard documents or software vendors. The protocol operates on a client-server mode; the clients are either user agents (browsers) or web robots (e.g., Google's indexing robot). When a client sends a request to the server, the stateless nature of HTTP implies that the request will not necessarily be related to prior or future requests. The server waits for a complete request, then sends a complete response. For example,

```
GET /aya2003/daily/20030715/thumbnails/009.jpg HTTP/1.1
Accept: */*
Referer: http://www.aya.org/aya2003/daily/20030715/
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5;
    Windows NT 5.0; (R1 1.3))
Host: www.aya.org
Connection: Keep-Alive
Cookie: PHPSESSID=8ce5ac388fc6c1f07d213df818f5a2e9
Extension: Security/Remote-Passphrase
```

Figure 2.1: An example HTTP request from a user agent, Internet Explorer version 5.5 from Microsoft. One line was broken to allow it to fit on the page.

```
GET /~bill/album/bill/petroglyphs_02/page1.html HTTP/1.0
Host: www.cs.unm.edu
User-Agent: WebVac (webmaster@pita.stanford.edu WebVac.org)
From: webmaster@pita.stanford.edu
```

Figure 2.2: An example HTTP request from a web robot agent, Stanford University's WebVac.

Figure 2.1 shows a sample request from a user agent, and Figure 2.2 shows one from a web robot.

The requested resource path can be a file name, but the interpretation is up to the web server. Some paths are intended to be interpreted by programs, and RFC 3875 [219] describes the common use of common gateway interface (CGI) scripts. One of the primary purposes of scripts is to provide a more dynamic experience for the user; therefore, a method for passing parameters to the script is provided. Normally, the path to the script is part of the requested resource path, the parameters follow a `?` in the path, and are of the form *`key=value`*. Figure 2.3 shows an example.

In order to allow a user to customize their view of a web site, several HTTP header lines exist; examples of these lines appear in Figures 2.1, 2.2, and 2.3. For example, the `Accept-Language` header line allows the user to specify their preferred language(s). Some web servers will use this information to present a language-specific version of the

```
GET /˜midhun/gallery/view.php?gid=2&phid=64 HTTP/1.1
Host: cs.unm.edu
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1;
    rv:1.7.3) Gecko/20040913 Firefox/0.10
Accept: text/xml,application/xml,application/xhtml+xml,
    text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Referer: http://cs.unm.edu/˜midhun/gallery/view.php?
    gid=2&phid=63
Cookie: PHPSESSID=383ca0237f3fbeecc253a988d99d8f70
```

Figure 2.3: An example HTTP request calling a CGI script and passing parameters. Three lines have been broken to allow them to fit on the page.

site. These headers are usually optional, and a server is free to ignore them.

2.4.2 Structure in the HTTP request

While a HTTP request is a string, it is also structured. Some portions of the structure vary more than others. For example, the method for the request is the first word in a request. The vast majority of requests use `GET`, but other methods, such as `HEAD` and `POST` exist, and extensions to the HTTP standard define more. For example, WEBDAV [97]) defines `PROPFIND`. Only a web server supporting this extension would ever see this method in non-attack traffic; other web servers will generate an invalid method error. As a second example, the resource path usually represents a file in the filesystem. Since all widespread operating systems use a tree-structured filesystem, a tree is likely to be a good representation for this part of the request.

On the other hand, some areas of the request vary more. For example, the `Referer` header provides the URI for the web page with a link to the requested page. When a user searches for a page, the search parameters are included in the search engine's URI in the `Referer` field. The diversity of search engine referrers is thus limited only by the

```
GET /~aaron/.../newmexico/BuddhistSprings_3.jpg HTTP/1.1
Accept: */*
-------: ----:-------------------------------------------
    -----------------------
---------------------
Accept-Language: en-us
---------------: ----- -------
If-Modified-Since: Sun, 23 Mar 2003 07:37:28 GMT
If-None-Match: "3faffe-a52f-3e7d6438"
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; Windows NT
    5.1; SV1; .NET CLR 1.1.4322)
Host: www.cs.unm.edu
Connection: Keep-Alive
```

Figure 2.4: An example HTTP request that passed through a proxy on its way from the client to the server. The proxy replaced some of the data by -s. Two lines have been broken to allow it to fit on the page, and the ... in the path replaced two directory names for the same reason.

different orderings of the different search terms that can be used to find the page on the different search engines. In general, the `Referer` values are limited only by the pages on the web with a link to the requested page; remember that remote web administrators control their content, so in effect the diversity is limited to the imagination of all of the web content developers who might want to link to the requested page. As another example, a portion of the HTTP protocol was designed to allow a web proxy to query the status of a resource before requesting it. To illustrate this, Figure 2.4 contains a sample request that passed through a proxy. Note the date on the `If-Modified-Since` line and the `If-None-Match` header containing a hash value. These hashes are designed to avoid collisions, and therefore should be unique.

2.4.3 Potential sources of diversity between web sites

Different web sites use different servers and provide different content. This diversity between web sites means that if an IDS can avoid overgeneralizing, it might distinguish between requests for different web sites. As will be shown in Section 5.1.1, the HTTP requests to different web sites are sufficiently distinct under some measures that an anomaly detector trained for one site does not work well for others. If the IDS can distinguish between web sites, an attacker might be forced to craft a distinct attack for each web site, slowing the attack rate to one more manageable by humans.

The first examples of diversity are the resource paths are chosen by the web site developer(s). They reflect the diversity of the developers' organization strategies for a web site. A request for the file `/foo/bar/blah.jpg` makes no sense at a web site that does not even have a `/foo` directory.

Because HTTP is stateless, various methods of providing a (pseudo-) state exist. One of these methods is the `Cookie` header. A cookie is a value that the server gives to the client. When the client requests a new resource, it includes the values of relevant cookies. Among other uses, cookies can customize a web site[6]. If the cookie makes proper use of cryptography, it can be used to validate a claimed state and to ensure that the user follows a prescribed path through a web site (e.g., the checkout process at an e-commerce web site). Cookie values vary with:

- the software running the applications on the web site. For example, Java and PHP both have library functions for session management, but they use different cookies for this task.
- different library functions may set different cookies (for tracking different parts of the user's state).
- the developer of the web site can choose cookie names for custom parts of the web site.

Some cookies contain hashes (e.g., the `PHPSESSID` in Figure 2.3).

While a couple of web browsers dominate the market, the list of web clients is large. Wikipedia [9] lists 138 different browsers, not counting the microbrowsers used on phones and other mobile computing devices. Add to this the diversity of web robots (at least one per search engine as well as those supporting research or scanning for vulnerabilities), then multiply by the number of versions of each of these and the list of potential values

[6] For example, FedEx.com stores the user's country in a cookie; Google.com uses cookies to store user search preferences.

for the User-Agent header becomes substantial. However, many web sites cater to a focused set of people, and these people and the web robots visiting the site might therefore represent a small subset of this potential diversity.

Beyond the user agent diversity, users in different portions of the world have different preferred languages. For example, a web site in Chinese is more likely to receive requests from people whose preferred language is Chinese than a web server hosting a web site solely in Swahili.

Another source of potential diversity between web sites is the user's preferred file types. For example, users visiting a web site providing information for blind people will likely have their browser configured to request audio versions when the server has them available.

Few researchers have noticed or tried to exploit this diversity. Estévez-Tapiador et al. [86] noted that even the two web servers in the MIT Lincoln Labs data received requests with a different probability density function for the request length. Kruegel and Vigna [146] noticed the diversity between their test data sets, but did not exploit it.

2.4.4 HTTP attacks

Attacks have a high diversity in location of attack and appearance, and differ from normal requests. To this point, most attacks have focused on the resource path. A few have targeted other portions of the request, and it would be foolish to suggest that the code that processes other portions of the HTTP request is invulnerable. The diversity of attacks against HTTP is high, representing the diversity of bugs that programmers introduce into code. This diversity implies that knowing one attack provides no guidance about the structure of the next one. This section illustrates some of this diversity by presenting example attacks.

One famous attack in the resource path is Nimda [58]. This attack targeted a collection of bugs in Microsoft systems, and used the fact that the default configuration enabled the attacker to exploit the vulnerability. Figure 2.5 shows one example (of 16 in my attack database). Another attack in resource path is the beck attack [50], shown in Figure 2.6. This attack is contained in the MIT Lincoln Labs data.

```
GET /scripts/root.exe?/c+dir HTTP/1.0
Host: www
Connnection: close
```

Figure 2.5: Example Nimda attack.

```
GET ////////////////////////////////////////////////
////////////////////////////////////////////////////
////////////////////////////////////////////////////
////////////////////////////////////////////////////
////////////////////////////////////////////////////
////////////////////////////////////////////////////
////////////////////////////////////////////////////
////////////////////////////////////////////////////
////////////////////////////////////////////////////
////////////////////////////////////////////////////
////////////////////////////////////////////////////
////////////////////////////////////////////////////
////////////////////////////////////////////////////
////////////////////////////////////////////////////
////////////////////////////////////////////////////
```

Figure 2.6: The beck attack. The original had 790 `/`s; the extra `/`s have been deleted to save space.

Another interesting example is illustrated by a vulnerability in Apache that allowed inadvertent disclosure of information [52]; an example is given in Figure 2.7. When including the `?M=D` in the request, the attacker would bypass the index page provided by the web developer, listing all files, including those intended to be more private. Search engines such as Google used this bug to index pages that had no externally accessible links to them.

Of the attacks that target a portion of the request not in the resource path, Figure 2.8 shows Apache Sioux [54]. This attack repeats a non-standard header line to cause a denial of service. Another example is the Apache chunked transfer error [53] (Figure 2.9). This attack exploited the fact that Apache had an error when the client requested chunked encoding transfers. Whether or not a web server used this type of transfer depends on the

```
GET /~immsec/data/SM/CERT/?M=D HTTP/1.0
Host: www.cs.unm.edu
User-Agent: msnbot/0.3 (+http://search.msn.com/msnbot.htm)
Accept: text/html, text/plain, application/*
Accept-Encoding: identity;q=1.0
From: msnbot(at)microsoft.com
```

Figure 2.7: The Apache M=D information disclosure attack.

data it served and the clients it serviced.

An attack type that is growing as the number of custom web applications grows is against CGI scripts. An example of one of these attacks is shown in Figure 2.10—the NT Index Server Directory Traversal attack [51]. This example was generated by typing the URL into a browser, and therefore appears similar to normal requests from this same type of browser.

2.4.5 Difficulties for anomaly detection posed by HTTP

Existing IDS solutions for web servers are weak (at best). One reason is that HTTP is a difficult protocol for an IDS to handle. The challenges include the stateless nature of HTTP, the nonstationary nature of web servers, the fact that HTTP requests are variable length, the training data for anomaly detection is unbalanced, and training data with attacks removed is difficult to obtain. Additionally, because harmless attacks are part of normal Internet web traffic, an IDS should be able to tolerate them without producing alarms, yet still detect novel attacks. These problems eliminate many algorithms that have been successful elsewhere.

Since HTTP is stateless, the IDS cannot rely on relationships between requests, sequence-based or otherwise; indeed, most existing attacks are only one request long. In attacks against other services or at other layers of the International Standards Organization (ISO) standard model for networking [123], an attacker might check if a vulnerability is likely to exist before attempting to exploit it. My data sets show that probes for web server vulnerabilities rarely precede actual attacks. With the use of botnets[7] for attacks,

[7] A botnet is a collection of computers that an attacker has compromised and is using to conduct

```
GET / HTTP/1.1
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
User-Agent: sioux
```

Figure 2.8: The Apache Sioux attack. The original had 1000 `User-Agent:` lines; the extra have been deleted in the version presented here. This attack was extracted from the MIT Lincoln Labs data.

```
GET /x.html HTTP/1.1
Host: 192.168.x.x
Transfer-Encoding: chunked

80000000
Rapid 7
0
```

Figure 2.9: One of the Apache chunked transfer error attack variants. This attack is described in [40].

```
GET /iissamples/issamples/oop/qfullhit.htw?CiWebHitsFile=
    /../../winnt/system32/logfiles/w3svc1/ex000121.log&
    CiRestriction=none&CiHiliteType=Full HTTP/1.1
Host: www.i-pi.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US;
    rv:1.4.1) Gecko/20031114
Accept: text/xml,application/xml,application/xhtml+xml,
    text/html;q=0.9,text/plain;q=0.8,video/x-mng,image/png,
    image/jpeg,image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Figure 2.10: One example of an NT Index Server Directory Traversal attack. Three lines were broken to allow it to fit on the page.

the attacker and legitimate user might be using the same computer. Thus, a web-server anomaly detection method must be able to analyze isolated transaction requests and determine whether or not they represent an attack on the server.

Beyond the stateless nature of HTTP, web site content changes rapidly [205]. An e-commerce site adds new products and removes obsolete ones. A university department web site changes as the semesters pass, classes come and go, new schedules are released, etc. Blogs are updated as often as the blogger provides new essays, often on a daily basis. Results in Section 5.1.2 show that the cs.unm.edu web site changed over the data collection period. Changes in content imply changes in the HTTP requests from the client to the server, resulting in changes in web server behavior. In addition, new versions of existing web browsers and new web browsers and robots are frequently introduced. In order to be successful, any IDS must be able to cope with this ever-changing (nonstationary) environment.

The challenges described above restrict the algorithms that can be used. The IDS must be able to adapt as the web server content or software changes, and this adaptation must not require time-consuming retraining. A web server must be closely watched during the

the attack. These computers are often home machines where the Internet connection is a broadband connection such as cable or digital subscriber line (DSL).

training period to ensure that no successful attacks occur, otherwise the anomaly detection system learns harmful behavior as normal. This higher human cost affects systems that need periodic retraining more than systems that can adapt as the web site changes. Chapter 4 discusses which of the algorithms applied to HTTP are well-suited to nonstationary data, and the test results in Section 5.1.2 show the algorithms' accuracy under these circumstances.

Additionally, as shown in Chapter 5.1.1, web server content, applications, and clients differ from each other. An IDS protecting a web server should be able to exploit this diversity to eliminate the risk of widespread attacks. Forcing attackers to craft a custom attack for each web site would change the timescale of the attack from one that occurs in computer time to one that occurs in human time, allowing human administrators to intervene and reduce the damage.

Another property of HTTP requests that limits the choice of algorithms is that a HTTP request is not fixed length. The path of the requested resource varies, and most header lines in the request are optional and of variable length. Any learning algorithm that requires fixed-length input, such as a neural net [195], will not work.

Many algorithms (including some that have been used for anomaly detection elsewhere) require similar amounts of normal (non-attack with harmless attacks, or at least non-attack) and abnormal (attack) data. A collection of normal HTTP requests can be obtained by intercepting the requests the server receives. A similar sized data set for attacks does not exist[8].

As illustrated earlier in Section 2.4.4 and shown to be true for my test data sets in Section 5.1.3, the diversity in attacks exceeds the diversity in non-attack requests. Because of this diversity, a small database of attacks is unlikely to be useful in predicting the next attack against the server. Therefore, an anomaly detection system for web servers must recognize patterns based on a set of normal requests—this is the definition of a one-class learning problem. As was shown in Section 2.2, limiting the learning to one-class algorithms limits the possible algorithms.

[8] For example, my attack database contains 63 entries and is the largest open database of those reported in academic literature (Chapter 2). One of my training data sets contains 390,911 requests.

```
GET /error/HTTP_NOT_FOUND.html.var HTTP/1.1
Host: www.i-pi.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.6)
    Gecko/20040114
Accept: text/xml,application/xml,application/xhtml+xml,
    text/html;q=0.9,text/plain;q=0.8,image/png,image/jpeg,
    image/gif;q=0.2,*/*;q=0.1
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
```

Figure 2.11: An example of an attack exploiting the Apache Win32 .var file web path disclosure bug. Two lines have been broken to allow them to fit on the page. This attack was generated using the Firefox web browser, hence it is similar to other, browser-generated requests.

2.4.6 Generalization and HTTP

The correct amount of generalization is necessary for an anomaly detection system to be accurate (Section 1.1). As an example of overgeneralizing with HTTP, consider a measure using the length of a HTTP request. Attacks such as the one shown in Figure 2.11 can easily be within normal; this one is 432 characters, placing it within the mean $\pm\sigma$ for the filtered cs.unm.edu 2004-11-12 training data[9]. For this data set, the mean length is 343 characters and sigma is 126, leading to a one standard deviation range of 217 to 469.

As an example of undergeneralizing, consider a simple anomaly detector that memorizes what it sees and flags anything not in this set as abnormal. This algorithm would undergeneralize whenever the training data is a proper subset of the normal data, for example when the set of normal is unknown or infinite. As a concrete example, this algorithm would undergeneralize on HTTP requests, because some proxies query to see if a page has been modified before requesting it (for one example, see Figure 2.4). These queries contain dates, and as time passes, the dates will change. Therefore, this simple anomaly detector would incorrectly flag normal requests as abnormal due to undergeneralizing.

[9] This data set is described in Section 4.3.

A particular problem for anomaly detection systems is mixed high- and low-variability data. If the algorithm treats each field uniformly, then it might undergeneralize high-variability data, overgeneralize low variability data, or both. The issue with many real systems, such as HTTP, is that different parts of the data have different levels of variability. For example, the header line keys such as `Host` and its values change little—the key introduces the host names that are used to describe the web server, presumably a small, finite set. Similarly, `Connection:` has two normal values, `keep-alive` and `close`. For these portions of the HTTP request, memorizing the values regularly seen is likely to be accurate without consuming a large amount of memory.

On the other hand, some of the HTTP request is more variable. Fields such as `If-None-Match:` include hashes, `If-Modified-Since:` contains dates, and cookies containing session identification strings used by languages such as PHP, Java, and ASP.NET are cryptographic hashes.

For an anomaly detection system working with HTTP, the model needs to have the resolution to be able to distinguish the different parts of the data. The algorithm must be able to apply different levels of generalization to different portions of the data.

2.5 Summary

Intrusion detection has existed since the early 1970s. The first attempts focused on insiders and host-based intrusion detection. When networks became more common, network-based IDSs at the various levels of the ISO protocol layer were developed. Unfortunately, for Internet servers such as web servers, these approaches are insufficient. To try to remedy the deficiency, researchers investigated intrusion detection at the HTTP layer.

Researchers have developed three architectures for intrusion detection: signature detection, specification, and anomaly detection. No general theory of anomaly detection exists, resulting in a need for rigorous testing to determine which systems perform best, and under what circumstances. Unfortunately, good testing is rare because it is difficult.

Good testing requires data representative of the protected web servers and data representative of attacks. The most commonly-used data sets from MIT Lincoln Laboratories are not representative of today's web servers, and they contain only four HTTP attacks. Some researchers, noticing this problem, obtained their own data. Unfortunately, these data are not publicly available, hindering direct comparisons between IDS systems or algorithms.

Of the three IDS architectures, this dissertation focuses on anomaly detection. Anomaly detection systems have the potential to detect novel attacks, but they sometimes suffer from high false positive rates. Approaches to solve this problem include: trying other algorithms, merging data from multiple, presumably independent sensors, and grouping putative attacks into classes. Research presented in this dissertation provides better guidance about how false positives can be reduced by increasing generalization, but only in portions of the data so that the system will be less likely to also miss attacks.

Chapters 5 and 6 show that controlling generalization is necessary for an accurate IDS. However, only one research group [168] has investigated the relationship between generalization and detector accuracy, and interestingly, their work contradicts results reported by other researchers.

This dissertation uses anomaly detection on HTTP requests sent to web servers to investigate anomaly detection system accuracy. HTTP is a stateless protocol that contains structure that can be useful in separating high- and low-variability portions of the request. The HTTP requests exhibit a diversity between web sites. Few researchers have noticed and none have attempted to exploit this diversity to slow attackers.

HTTP attacks have an even higher level of diversity than normal requests, indicating that knowledge of one attack will be of limited use in predicting other attacks. HTTP also presents other challenges for developers of IDSs. The challenges include the stateless nature of HTTP, the nonstationary nature of web servers, the fact that the HTTP request is variable length, the training data for an anomaly detection is unbalanced, and training data with attacks removed is difficult to obtain. Additionally, since Internet-connected web servers are routinely attacked with harmless attacks that are not (and might never have been) a threat to the server, the anomaly detection system should not produce alarms for these attacks. This requirement places additional stress on anomaly detection systems.

Only one of the previous anomaly detection systems for web servers used the complete HTTP request. The rest focused on the requested resource path, the first n bytes, or, in one case, part of the requested path. If the IDS does not make use of the whole HTTP request, it will miss some attacks. The work presented in this dissertation is one of the first to use the complete HTTP request.

Chapter 3

Algorithms for HTTP anomaly detection

As mentioned in Chapter 2, several algorithms have been proposed for anomaly detection with HTTP. I implemented those that had been tested on HTTP when I began the research. I also implemented two algorithms that seemed promising for the HTTP domain: n-grams and DFA induction. I re-implemented previously-proposed algorithms for three reasons:

- I wanted a testbed where I could see if the results described in the papers were reproducible.
- I wanted to explore the algorithms in ways not originally intended by the authors.
- I wanted a consistent and comparable environment for all algorithms.

This chapter describes the algorithms, including any implementation issues I faced, decisions I made in the event of unclear descriptions in the papers and differences between the description in the paper, and my implementation. Because HTTP is a nonstationary environment (Section 2.4.5), if the author(s) did not address this issue explicitly, I discuss potential modifications that would to enable it to work with nonstationary data. For each algorithm, I also discuss the mechanism by which the algorithms generalize and if they distinguish between various parts of the HTTP request. I use this analysis to predict if the will algorithm over- or undergeneralize, and how it will perform when tested on data containing attacks.

3.1 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 [146] 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:

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

Kruegel and Vigna did not state how they handled the case $l = \mu$; in my testing, this situation never arose. A likely explanation for $l \neq \mu$ is that, for real data, l is an integer and μ is probably not.

In my implementation of this method, the algorithm can be used either as part of the linear combination (Section 3.7) or as a standalone algorithm.

The length measure generalizes by classifying any string with length is near the mean as normal. When trained on filtered data, this measure is likely to detect attacks such as cross-site scripting and buffer overflows, because they are typically longer than normal requests. This measure will likely miss attacks similar in length to normal requests. The length measure does not distinguish between parts of a request. This means it is unlikely to perform well when trained on data containing attacks.

For nonstationary data, μ and σ^2 could be updated via Knuth's observation that given a new data point x_{N+1}, a new μ' can be generated from μ via:

$$\mu' = \mu + \frac{x_{N+1} - \mu}{N + 1}$$

For nonstationary data, it might be possible to weight the new observations in such a way to diminish the importance of older values. Kruegel and Vigna did not mention using the algorithm with nonstationary data; I did not investigate using this algorithm in a manner adapted for nonstationary data.

3.2 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 [263] used a character distribution metric on similarly-sized packets. Kruegel and Vigna [146] used a character distribution as one of six tests in the linear combination (Section 3.7).

Character distributions generalize by accepting any string having a distribution of characters matching the learned distribution. When trained on filtered data, these measures are likely to detect buffer overflow, cross-site scripting, and attacks such as beck (Figure 2.6). These measures are unable to distinguish between parts of a request, so they are unlikely to perform well when trained on data containing attacks.

Mahalanobis distance

Wang and Stolfo [263] measured the Mahalanobis distance, d, between two distributions. Given the new feature vector x and an averaged feature vector $\overline{y}$, then

$$d^2(x, \overline{y}) = (x - \overline{y})^T C^{-1} (x - \overline{y})$$

where C^{-1} is the inverse of the covariance matrix, with $C_{ij} = \text{Cov}(y_i, y_j)$. Wang and Stolfo assumed that the bytes in a request are statistically independent. This assumption means that the covariance matrix is diagonal, and the diagonal elements give variance of each byte. Next, to speed up the calculation, they derived a measure they called *simplified Mahalanobis distance*:

$$d(x, \overline{y}) = \sum_{i=0}^{n-1} \frac{\mid x_i - \overline{y_i} \mid}{\overline{\sigma_i}}$$

n is 256 for the ASCII character set. It is possible that the standard deviation $\overline{\sigma_i}$ could be 0, implying an infinite distance. To avoid this situation, they used α as a smoothing factor:

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

For α, Wang and Stolfo did not specify a value although they discussed determining it automatically (without specifying how). I used 0.001, but this value is a parameter and can be changed at run time. By setting the distance threshold to 256, they allowed a fluctuation of one standard deviation. In effect, their system considers rare distributions anomalous; this explains why when they tested the system on the training data, it did not have perfect performance.

For the actual implementation, I used Knuth's method (described in Section 3.1). Noting that the standard deviation is the square root of the variance and the definition of variance using expected values $E()$ can be written as:

$$\text{Var}(X) = E[X - E(X^2)] = E(X^2) - [E(X)]^2$$

By keeping track of the average of the frequencies and the square of the frequencies, calculating the standard deviation is fast. This approach means that new observations can be incorporated into the measure, a method for handling nonstationary data. However, they provided no way of "forgetting" old information, although it might be possible to weight the new observations more than the $\overline{x}$. Neither Wang and Stolfo nor I pursued this idea.

For my implementation of this distance metric, given an instance, I calculate relative character frequencies ($f(c)$ for character c) of a request. Using Knuth's method, I record the mean and mean squared for each character seen from the training requests (two 256-element arrays: $\overline{x_c}$ and $\overline{\sigma_c}$). Then the distance is calculated:

$$d = \sum_{c=0}^{255} \frac{\mid f(c) - \overline{x_c} \mid}{\overline{\sigma_c} + \alpha}$$

Since d is finite, I map it into $[0, 1]$ via:

$$s = \begin{cases} 1 & d \in [0, 1] \\ \frac{1}{\ln{(d+e-1)}} & d > 1 \end{cases}$$

Note that e is the base of the natural logarithms. The mapping for values of $d \leq 1$ is justified because the larger the distance, the more abnormal the data item. Experience shows that requests from the training data set have a distance $d > 1$, so the few requests with a small distance are clearly close to the learned distribution.

Wang and Stolfo correlated packet length with character frequencies. My data consists only of the data at the application layer; the packets containing the data were not stored. Therefore, I apply this method to the complete request.

χ^2 of idealized character distribution

As one of six tests, Kruegel and Vigna [146] 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) 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.

Kruegel and Vigna did not address nonstationarity explicitly, but, by using Knuth's method, new observations could be added to the $\mathcal{ICD}$. For forgetting older data, it might be possible to weight the new observations more than the older ones.

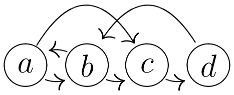


Figure 3.1: Example directed graph induced by `abcd` and `dbac`.

3.3 Ordering of parameters

For other measures in their system, Kruegel and Vigna [146] observed that since CGI parameters are set by one or more programs, the normal order of the parameters is often fixed. If a human generates the path, the order could be different, and they presumed this change indicated a potential attack.

For learning the order of CGI parameters, they built a directed graph representing the order of the attribute values. Testing was then a simple matter of attempting to traverse the graph, with 0 or 1 indicating success.

My implementation of this algorithm can be used either as part of the linear combination (Section 3.7) or for more generic data items. However, I did not perform tests with this algorithm as a stand-alone algorithm because it was not clear how to map an entire HTTP request into an order relationship that differed from the DFA and n-gram methods. Additionally, this algorithm considers only the order of the parameters, and ignores the values (other measures in the linear combination use values). Testing of other algorithms such as the DFA showed that token values were necessary for reasonable accuracy.

Kruegel and Vigna did not discuss nonstationary data. It should be possible to add additional edges into the graph, and by keeping track of usage for the edges, lesser-used (and therefore presumably older) edges could be dropped.

This algorithm generalizes by allowing orderings not originally seen in the training data. For example, if the training data consisted of `abcd` and `dbac`, the directed graph would be as in Figure 3.1, and `acdb`, `bacd`, and `cdba` are also in the directed graph, and hence be considered normal. Depending on the data tokenization method, this algorithm has the potential to distinguish regions of the request.

3.4 Presence or absence of parameters

Kruegel and Vigna [146] noted that CGI parameters are provided by a program and therefore the same parameters are provided, even if one or more have no value. The result is a regularity in the number, name, and order of the parameters. Kruegel and Vigna's 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.

The presence or absence algorithm requires the CGI program path in addition to the parameters. This requirement for additional information means it does not meet the algorithm interface standard for the test object. I used it only as part the linear combination (Section 3.7).

The generalization of this algorithm is limited. If the program `/foo/bar` was observed at various times with individual parameters `a`, `b`, and `c`, then this measure would also consider normal any combination of these parameters. It would be up to the ordering of parameters algorithm to identify that the combination had not been seen previously.

To handle nonstationary data, this algorithm could be modified to record the frequency of remembered parameters and paths. Non-used paths or parameter lists could be dropped and newer ones added.

3.5 Enumerated or random parameter values

Similar to the presence and absence test, Kruegel and Vigna also noted that some CGI parameter values are selected from a finite set (enumerated), and others are effectively random. In the training phase, given instance number i, the anomaly detection algorithm calculates two functions:

$$f(i) = i$$

$$g(i) = \begin{cases} g(i-1)+1 & \text{if the value is new} \\ g(i-1)-1 & \text{if the value is not new} \\ 0 & \text{if } i = 0 \end{cases}$$

Note that f is strictly increasing, and g increases only when new values appear in the training data. The correlation parameter ρ is

$$\rho = \frac{\mathrm{Cov}(f, g)}{\sqrt{\mathrm{Var}(f)\mathrm{Var}(g)}}$$

If $\rho < 0$ then f and g are negatively correlated, and the values are presumed to be enumerated, and if $\rho > 0$ then the values are assumed to be random. Kruegel and Vigna did not mention the case $\rho = 0$; in my implementation, random is presumed from $\rho \geq 0$. If the values are enumerated, they are remembered for testing.

When testing, if the training data indicated the values were random, then the similarity is 1. If the training indicated that the data comes from a set (i.e., it is enumerated) and the value is in the learned set, the similarity is also 1. If the value is not in the learned set, the return value is 0.

This test is not easily modified for nonstationary data.

If the algorithm determines a parameter value is enumerated, then it performs no generalization. On the other hand, if the parameter value is random, everything is accepted. The algorithm therefore might be subject to both under- and overgeneralization problems, depending on the specific CGI program. For example, if one of the parameters was a user name, then the list of valid users would be enumerated (assuming a sufficiently large training set). Any time a new user was added, the system would flag references to this user as abnormal. On the other hand, if a parameter was deemed to be random, then buffer overflow and cross site scripting attacks are two of the attacks that would be missed.

It was unclear how to apply this measure to the complete HTTP request. Using the HTTP header keys as parameters and the HTTP header values as values resulted in the test system running out of memory during training. Therefore, I used it only as a part of the linear combination (Section 3.7).

3.6 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 [245] and Stolcke and Omohundro [244]. Warrender et al. noted that building a generalized Markov model is $\mathcal{O}(n^2)$ [264].

Markov models have been shown to be an effective but time-consuming algorithm for intrusion detection [264]. Kruegel and Vigna [146] used a Markov model as a portion of the IDS for protecting web servers, but after noting that the probability of any given 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.

My Markov model implementation is a modification of the DFA algorithm described in Section 3.9. 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.

It is not clear how to modify the Markov model for nonstationary data using the traditional method for inducing the model. In my implementation, the same update procedure for the DFA can be applied to the Markov model.

Generalization in the Markov model occurs via the state merging operations, that might allow combinations not seen in the training data. This generalization is limited, and novel values will result in a probability of 0. Therefore, in a nonstationary environment, the Markov model is likely to perform poorly.

When trained on tokens, the Markov model can distinguish between regions of the input data, and hence it should be able to distinguish between attacks in different portions of the test data instances.

3.7 Linear combination

The system developed by Kruegel and Vigna [146] was limited to HTTP CGI requests, and consisted of a linear combination of the following algorithms:

Length for CGI parameter values.

Character distribution for CGI parameter values.

Markov model for learning the structure of parameter value strings. They mapped all lowercase letters to a lowercase letter class, and used the Markov model as a DFA to indicate only the ability to generate the string and not the probability of generating it.

Enumerated or random parameter values for CGI parameter values.

Presence or absence of parameters.

Order of parameters.

The threshold for normal for each algorithm was determined dynamically, chosen to be 10% above the highest value obtained. Calculating the threshold requires a second pass over the training data. This two-pass approach makes handling nonstationary data difficult.

For testing, each algorithm was equally weighted and the system produced a binary normal/abnormal signal. My implementation of this algorithm returns the average of the six measures.

The generalization performed by this measure is the generalization performed by the various algorithms. My implementation accepts only those data instances that are accepted by most of the individual algorithms. The method for combination controls how important the individual algorithms are. If the combination was an AND operation, then any one algorithm could cause the rejection of a data instance. On the other hand, if the combination was an OR operation, then acceptance by any algorithm would result in acceptance by the combination.

Because some algorithms undergeneralize and others overgeneralize, the algorithm for combination affects how the combination generalizes. In the AND combination, undergeneralization of a single algorithm results in the combination undergeneralizing. However, in the OR combination, overgeneralization by a single algorithm results in the combination overgeneralizing. Using the average helps reduce the problems caused by a single algorithm under- or overgeneralizing, assuming the remaining algorithms generalize properly.

The ability to distinguish regions is controlled by the various algorithms and their ability to distinguish the regions.

3.8 n-grams

An n-gram [57] is 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`.

When used in intrusion detection, the n-grams in the test instance are compared to those in the training data. The similarity measure might take into account the frequency of occurrence of the n-grams in the training data, or it might simply consider 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 results in this dissertation are based on the second version. Because the n-gram includes context for a given token, it also encodes the structure of the HTTP request, although in a different manner from the DFA and Markov Model. Therefore, the n-grams can distinguish between regions in a HTTP request.

The n-gram algorithm can use either tokens or strings from the data source. Early testing showed poor accuracy on strings, so the results reported in this dissertation uses tokens.

The n-grams induce a directed graph. The generalization therefore is similar to that described for the order of parameters algorithm (Section 3.3).

For nonstationary data, the algorithm can add n-grams to the set. By keeping track of usage frequency of each n-gram, lesser-used ones can be deleted.

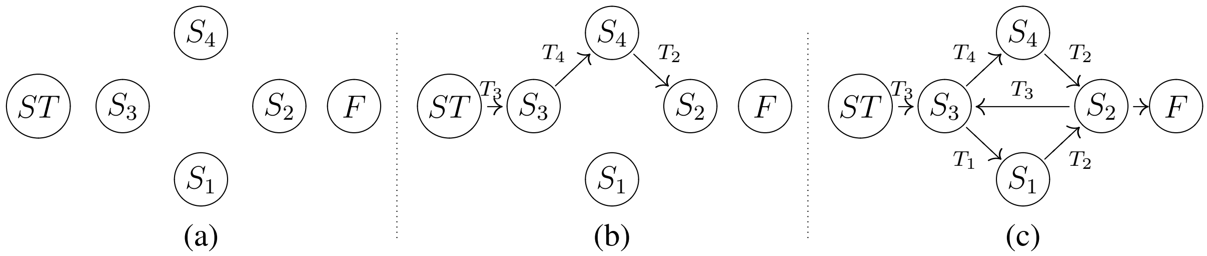
3.9 DFA

Although the formal grammar that describes HTTP in standards documents [88] is context-free, I discovered that in practice a given web server uses only a subset of the full HTTP language. As a result, most HTTP requests can be described using the simpler DFA representation. Because the DFA encodes the relationship of various potions of the data to each other, it can distinguish between areas in the HTTP request. I implemented a DFA induction algorithm known as the *Burge DFA induction algorithm*, and added heuristics that help keep the learned DFA tractable and able to process nonstationary data. These are described below.

The Burge DFA induction algorithm is a one-pass, $O(nm)$ 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, and as I will describe, the resulting DFA can be modified easily to deal with a nonstationary environment.

To model a web request, I construct an initial DFA as follows. First, let $\Sigma = \{T_1, \ldots, T_n\}$ be the set of n unique tokens in the HTTP request, and let $L = (l_1, \ldots, l_t)$ with $l_i \in \Sigma$ be the series of t chronologically ordered tokens from the HTTP request, with `FINISH` added in as a special token to indicate the end of the request. Let $G = (S, A)$ be a DFA with states S and transitions A. $S = \{S_{\mathit{START}}, S_1, \ldots, S_n, S_{\mathit{FINISH}}\}$ where states $S_1, \ldots, S_n$ have a one-to-one correspondence with tokens $T_1, \ldots, T_n$, and S_{START} and S_{FINISH} are additional states. $E(T)$ is a function that returns the state to which the token T will cause a transition. $A = \{A_{i,j}\}$ where $A_{i,j}$ indicates a transition labeled T_j between states S_i and S_j.

Given these definitions, the algorithm proceeds as follows:



(a) (b) (c)

Figure 3.2: The DFA induced by the Burge algorithm for the series of tokens $T_3, T_4, T_2, T_3, T_1, T_2$. a) The initial empty DFA with one state for each token. The *ST* and *F* nodes correspond to the S_{START} and S_{FINISH} states. b) The state of the DFA after the tokens T_3, T_4, T_2 have been read. c) The final state of the DFA after the entire series of tokens has been read in. Note that all states corresponding to consecutive tokens in the HTTP request are connected with a transition.

```
1. Set the current state $C = S_{\mathit{START}}, A = \emptyset$.
2. For $j = 1$ to $t$ :
    (a) If $A_{C,E(l_j)} \notin A$ then $A \leftarrow A \cup A_{C,E(l_j)}$
    (b) $C \leftarrow E(l_j)$
3. $A \leftarrow A \cup A_{C,S_{\mathit{FINISH}}}$
```

A DFA G is constructed with one state for each unique token in the HTTP request, as well as the two additional states S_{START} and S_{FINISH}. At the start of the algorithm, no transitions are present in the DFA and a current state C is set to the S_{START} state. Then, in step 2, tokens are sequentially processed from L. In step 2a, if no existing transition exists between the current state and the state corresponding to the current token, the transition is created and the new transition is labeled with the current token. C is then updated to be the state corresponding to the current token. After all the tokens have been processed, a transition from C to S_{FINISH} is created. Figure 3.2 shows an example. At this point, the DFA model has one state corresponding to each unique token present in the HTTP request, and the DFA has a transition between any two states that were seen consecutively in the request. Each transition is labeled with the token corresponding to the destination state.

Additional requests are processed by running the algorithm with the tokens from the new request, adding nodes as needed for new tokens and adding edges to the DFA as

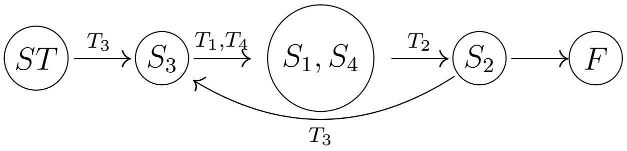


Figure 3.3: Compressed version of the DFA in Figure 3.2c. States that have identical sources and destinations (S_1 and S_4) are compressed into the same state ("S_1, S_4"). Tokens that originally caused a transition into one of the uncompressed states cause transitions into the compressed state.

described above. As the learning proceeds, the compacting processes described below are regularly performed.

Compacting and Generalizing the DFA model

In practice, the DFA induction algorithm described above leads to large, complex DFAs that potentially could grow without bound in dynamic environments with perpetually novel HTTP requests. I use two techniques to manage this complexity, one that compacts an existing model and one that adds states and transitions incrementally and "forgets" structures that have not been used.

To reduce DFA size, the algorithm searches at regular intervals for nodes that have the same source and destination (sorting the nodes by source and destination can allow this comparison to run in $O(n \log n)$ time). These nodes are combined into a single node, as illustrated in Figure 3.3.

This compression is a form of generalization. For example, suppose that during learning the DFA was compressed, producing Figure 3.3. Then, as learning continued, the T_1 token was observed leading to the compressed state (S_1, S_4). Given the DFA's current topology, the next expected token would be T_2. However, if a new token, T_5, was observed before the expected T_2 token, the DFA learning algorithm would insert a new state, S_5, into the DFA (Figure 3.4). This new topology is a generalization of the observed token sequence because either a T_1 or T_4 token followed by a T_5 token will lead to the new S_5 state (Figure 3.5). Thus, modifying the topology of a compressed DFA allows the resulting DFA to generalize from any of the constituents of a compressed node to other similar but unobserved behavior involving the node's other constituents.

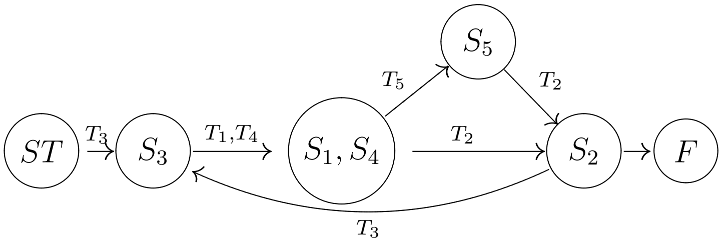


Figure 3.4: Compressed DFA from Figure 3.3 after additional learning.

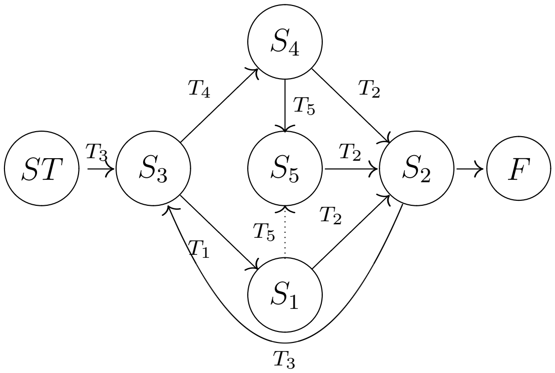


Figure 3.5: The DFA in Figure 3.4 without compression. The dotted link between states S_1 and S_5 indicates what is generalized when learning on the compressed DFA.

Without compression before the T_5 token was observed, this generalization would not occur because once the T_5 token is observed, the S_1 and S_4 states can no longer be merged. However, the method might miss some generalizations by not compressing the DFA before generalizable behavior is observed. This limitation could be addressed by counting the number of times certain links have been observed and compressing all states that differ by a small number of incoming and outgoing link counts. It is not clear that this generalization is valid. However, without regular compression of the nodes, the DFA grew too large for memory on some test data. Future work includes further investigation of this generalization.

3.9.1 Determining similarity between a request and the DFA

A DFA by itself is simply a language acceptor; however, I expect some variation in normal behavior (Section 2.4.5) not covered by the generalization algorithms. In this section I define a distance measure for comparing a DFA model with a web request to determine how anomalous it is. With this measure, I can tune the relative rate of true positives to false positives.

During testing, the DFA model is used differently from a traditional DFA. When a token is processed that is illegal according to the DFA, a "missed token" event is recorded and an attempt is made to resynchronize. For example, in Figure 3.3, suppose the current state was S_1, S_4 and the next token was T_3. If the edge corresponding to the next token exists elsewhere in the DFA, then the tester could transition to the edge's destination. Continuing the example, the new current state would be S_3, and a missed-token event would be recorded. If no such edge exists, a second missed-token event is recorded, and the tester moves to the next token, again attempting to resynchronize. The number of missed tokens provides an estimate of how anomalous the request is with respect to the DFA. The similarity s between a HTTP request and the DFA is calculated by:

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

The similarity measure reflects the changes that would have to be made to the DFA for it to accept the request (i.e., for each missed token a new transition would have to be added). Note that the significance of a single missed token by this measure depends upon the total number of tokens in the request. One benefit of this sensitivity is that more complex requests (i.e., those specifying many of the HTTP header options) have more room for slight changes and can still be accepted as normal. A measure that is less sensitive to the token count is likely to have a higher false positive rate due to the common, benign variations in complex requests.

Unlike Markov models, the DFA does not encode probabilities for each link in the state transition table. Rarely accessed parts of a web site (e.g., pages "about this web site") or rarely used configurations of web clients can thus be tolerated. Of course, the DFA itself is induced from an observed sample of requests, and the more common examples are more likely to be present in the sample.

The distance measure is a form of generalization. If an attacker generates an attack containing a large enough number of normal tokens, she can hide her attack from the DFA because in this circumstance it would be overgeneralizing.

Nonstationarity

To track web sites that change over time, the learned model needs to update itself automatically. When a HTTP request arrives that is not captured by the current DFA but has a high similarity value, the DFA is modified to accommodate the request. The threshold for this operation is controlled by a parameter specified by the system administrator; the threshold would normally be set near 1.0, so that only requests that are very similar to the DFA are learned. For the results presented in this dissertation, the value is set to 0.9 unless otherwise stated.

To detect unused edges, a counter is updated every time an edge is traversed. The counters are used during periodic pruning sweeps when infrequently used edges are deleted. After each sweep, all counters are reset to 0. A parameter controls how aggressive the pruning pass is, and if it is set to 0, only edges that have not been used since the last pruning pass are deleted. This allows recently added edges to survive for one full interval between prunings.

3.10 Targeted generalization heuristics

I implemented several heuristics that increase the generalization for specific locations in the request. The parser checks to see if the high-variability area values are well formed, and replaces them with an indication of if the value met the standard. The list of heuristics relating to generalization is:

recognize hostnames Rather than have the IDS algorithm attempt to learn every host name in the Internet, the parser will validate the form of the hostname and ensure that it meets the Internet standard. The return value is whether or not the name is valid.

recognize IP addresses Similar to the host name validation, this option validates the form of IP addresses.

lookup hosts For the hostname validation, this option will cause a lookup in the domain name system (DNS) to confirm that the host name is a valid one.

recognize dates Since dates change every day, they present a problem for an IDS algorithm learning the structure of a request. This option causes the date to be validated as meeting the standard. Interestingly, even though the standard describes three accepted date formats, many clients use other variants. The parser recognizes all formats that appear in the test data.

recognize PHPSESSID and Entity Tags These are hashes, and one purpose of a hash is to be unique. Rather than have the IDS algorithm attempt to learn every hash, these options simply have the parser validate the form (character set and length) of the hash and return if the hash is valid or not. Additional hashes (e.g., JSESSIONID) exist, and future work includes capturing and validating these hashes as well.

Each of these heuristics can be individually turned on or off for a given test run.

Experience working with the algorithms led to the development of additional heuristics that affected generalization in one direction or another.

3.10.1 File name heuristics

In running a test with 6-grams, they ran out of memory. Looking at the 6-grams discovered showed that the algorithm was memorizing file types. Most requests to a web server are for files of a few types (e.g., HTML files, JPEG files, etc). Instead of learning every file on the system—data that change as web site contents change—the parser attempts to identify the file type for the last component of the path based on the file name, and it will return this identification and not the file name. This is the **file types only** heuristic. Using this option reduced the size of the DFA, and the size of the collection of 6-grams.

However, both the 6-grams and DFA true positive rate fell. Inspecting the missed attacks showed that the IDS algorithms were missing buffer overflow attacks in the file name. Therefore, a slight reduction in generalization was called for, the **file name lengths** heuristic that applies when the file is an unknown type; in this event, the parser returns the length instead of the file name.

3.10.2 Floating-point numbers

Manually inspecting the DFA showed that the part of the DFA associated with q-values showed signs of memorizing floating point values. A q-value is a floating point value (in [0,1]) used in negotiations between a client and server for features such as preferred languages and file types. The heuristic **recognize q-values** results in the parser validating the value as meeting the standard; instead of returning the value, the results of the validation (OK or not OK) is returned. The result is slightly increased generalization.

3.10.3 Upper and lower case

The HTTP standard states that the HTTP header names (e.g., `User-Agent`) are case-insensitive. Because of the standard, different user agents use different capitalization schemes for the same header. The parser treats all keys as case-insensitive per the standard.

The **lowercase only** heuristic maps the request (both names and values) to lowercase, reducing the variants that an IDS algorithm must learn as normal. This heuristic has the effect of increasing generalization. This type of generalization would not easily be discovered by the automatic discovery process, and the difference in accuracy is likely to be small.

3.10.4 Email addresses

Email addresses are part of some HTTP header values. The host name part of the address can be verified (see the heuristics recognize hostnames, recognize IP addresses, and lookup hosts above). However, short of attempting to send email, no user name validation can be performed. The **email user length only** heuristic causes the parser to return the length of the user name and not the name itself. The length was important; otherwise buffer-overflow attacks in the user name are a possibility that the algorithms would miss.

3.10.5 Deleting unusual lines

Many false positives are caused by HTTP requests with unusual lines. Because these lines are not critical for the web server to identify the requested resource, the system tries deleting the following HTTP header lines, one at a time: `Referer`, `Cookie`, `Accept-Language`, `Accept-Charset`, and `Accept`. If, after deleting a line, the request passes the similarity test, then it is accepted and processed without the anomalous header lines. This is the **try alternates** heuristic.

With the idea that the IDS protecting the web server would be implemented as a proxy, this deleting of a line might protect the web server from an attack delivered through that portion of the HTTP request. Aside from cookies, the worst potential impact on a user is that a web client might receive the default version of a web page instead of one customized to its preferred language, character set, or file format.

Deleting cookies is potentially more serious, because they might encode state (e.g., the PHP session identifier cookie) that is required for proper operation of the web site. Therefore, deleting a cookie could interfere with the user's ability to visit the web site. Some web clients send cookies to web sites that do not match the cookie's list of sites that may read and modify it. These unexpected cookies then cause the request to be identified as abnormal.

3.10.6 Grouping putative attacks

I added another heuristic after noticing a web robot that came online during testing, and was not observed in the training data[1]. The one robot was responsible for nearly 10,000 false positives because its request format was different from the requests in the training data. To account for such situations, this second heuristic groups putative attacks into classes. When a request is classified as being abnormal, it is compared to the DFAs in a set of DFAs representing putative attacks. If the anomalous request is similar enough (controlled by a parameter I call the "grouping threshold"), it is incorporated into the DFA representing a class of attacks. If it is not similar to any of the existing attack classes, it is used to start a DFA for a new class of putative attacks. The comparisons and additions to

[1] See Figure 2.2 for an example request from this robot.

Heuristic	**State**	**Heuristic**	**State**
email user length only	on	file name lengths	on
file types only	on	recognize Entity Tags	on
recognize PHPSESSID	on	lowercase only	on
lookup hosts	off	recognize dates	on
recognize hostnames	on	recognize IP addresses	on
recognize q-values	on	return values	on
try alternates	on	group possible attacks	off

Table 3.1: The default state of the parsing heuristics for the test runs.

DFAs are as previously described in Section 3.9. By grouping related unusual requests, an administrator could look at a single exemplar of the class and determine if it is acceptable. If so, the request could be added to the main DFA in the normal way, and future similar requests would be accepted as normal. This heuristic is **group possible attacks**. This heuristic is off by default except for the tests relating to it because it requires a different, much more time-consuming test methodology to generate ROC curves.

3.10.7 Add alternates

Some algorithms can add new instances as they run to adapt to nonstationary data. This option enables adding new instances to the model, but only if the instance has a similarity value above the **abnormal threshold**. The default value for this threshold is 0.9 to avoid adding unusual instances as normal.

3.10.8 Default values of heuristics

As Section 5.5 will show, the heuristics have a dramatic effect on the performance of the algorithms. Table 3.1 presents the default parser heuristic settings. Unless otherwise specified, these settings are used for the test results in this dissertation.

Algorithm	**Generalization**
Length	overgeneralizes
Mahalanobis distance	overgeneralizes
χ^2 between character distributions	overgeneralizes
Markov model	undergeneralizes
Linear combination	both over- and undergeneralizes
n-grams	undergeneralizes
DFA	undergeneralizes

Table 3.2: Predictions about under- and overgeneralization of the algorithms for HTTP requests. The measures order of parameters, presence or absence of parameters, and enumerated or random parameters were not applied to complete requests, and therefore are not a part of this table.

3.11 Summary

My testing of algorithms included those proposed for use with HTTP requests at the time I began my research: Mahalanobis distance between character distributions; and a linear combination of length, χ^2 between idealized character distribution and the test distribution, a Markov model, order of parameters, presence or absence of parameters, and whether parameters are enumerated or random. I further investigated some of these algorithms individually. Additionally, I implemented two algorithms new to HTTP anomaly detection: n-grams and DFA induction. I also developed several heuristics that alter the generalization of the token-using algorithms. I implemented some of the algorithms in a manner more general than the original—several were originally applied only to CGI parameters; my implementation can work with the whole request as well.

Some, but not all, of the algorithms can be used with nonstationary data. The significance of this limitation will become clear with some of the results presented in Chapter 5.1.2.

Each of these algorithms has differences in how it generalizes, whether or not it can work with nonstationary data, and if it can distinguish between parts of a HTTP request. These differences allow me to make the predictions in Table 3.2 for when the algorithm is applied to a complete HTTP request. Later chapters in this dissertation will present test results to confirm or refute these predictions.

Chapter 4

Experimental setup

To perform rigorous tests of HTTP IDS algorithms, they must be tested under identical circumstances. This chapter describes the framework and the data used for testing. Section 4.1 gives an overview of the test framework. Some of the IDS algorithms require a list of tokens; other IDS algorithms work with a string representation of the HTTP request. The parser object provides either representation. Section 4.2 describes the parsing process and the difficulties that arise when working with real data (compared to data that is fully standards-compliant).

A requirement for comparing algorithms is to test them on identical data; this requires a collection of test data representative of data delivered to production web servers. Quality test data is difficult to obtain; organizations with the most interesting data typically consider it to be confidential. As a result, I collected my own data for algorithm testing from four web sites (two web servers). For one of the web sites, the attacks that occurred in the test data are available for the tests on data containing harmless attacks (Section 2.3.2 motivated this need). Section 4.3 describes these data in more detail.

The attack data needs to be representative of the broad range of attacks that exist today. As Section 2.3.2 showed, no such pool of attacks existed when I began my research. Therefore, I collected my own set of attacks. Collecting these data has its own challenges, and Section 4.4 discusses these challenges and describes my attack data.

4.1 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 I did not have to re-implement any of the surrounding support code. 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, I wrote an IDS algorithm object that uses *snort* signatures for checking HTTP requests.

To introduce the algorithm object, first note that an anomaly detection algorithm normally has two phases:

1. A training phase, in which the algorithm observes instances of behavior and generalizes from them. Some algorithms (but none of those I implemented) also have a supervised learning phase in which examples of known attacks are provided. Some algorithms (e.g., the linear combination) require two passes over the training data.

2. A testing phase, in which the algorithm is presented with an instance and determines the likelihood of this instance being an attack.

The algorithm object provides a method for each of these phases. In the training phase, the parser object returns individual instances that the algorithm uses to build a representation of normal. In the testing phase, the algorithm receives an instance and returns $s \in [0, 1]$, representing the similarity of the test instance to what the learned system expects. $s = 0$ for a completely novel instance, and $s = 1$ for an instance with no differences from the learned examples. Intermediate values represent intermediate similarity. In the original research descriptions, some of the algorithms used a threshold. To aid in determining the best threshold value, I report data showing how different thresholds perform. For the algorithms that originally returned a similarity value outside of $[0, 1]$, I mapped the result into this range.

```
GET /aya2003/daily/20030715/thumbnails/009.jpg HTTP/1.1
Referer: http://www.aya.org/aya2003/daily/20030715/
Accept-Language: en-us
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/4.0 (compatible; MSIE 5.5;
    Windows NT 5.0; (R1 1.3))
Host: www.aya.org
Cookie: PHPSESSID=8ce5ac388fc6c1f07d213df818f5a2e9
```

Figure 4.1: An example HTTP request. The `User-Agent` line has been broken to allow it to fit the page. In the original request, it was one line. The way the line was broken preserves it as a valid request.

4.2 Parsing HTTP

Some algorithms require that the data be tokenized. For these algorithms, I implemented a parser that breaks the HTTP request into tokens based on the those specified in the HTTP standard, RFC 2616 [88]. 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. Given the sample HTTP request in Figure 4.1, the parser produces the list of tokens (one per line) with values in Figure 4.2.

Instead of using tokens, some algorithms use a string representation for the request. This representation is available from the parser; for this case, the parser runs only the code that identifies a single request from a file potentially containing several (this code is a prerequisite for identifying the tokens).

The parsing process is complicated because some web browsers and many web robots do not comply with the HTTP standard [88]. This implies that any solution requiring software to parse the HTTP request must be more general than the standard[1]. For example, although the standard specifies three different formats for dates, my parser recognizes six. Another example is the following `Referer:` line:[2]

[1] This requirement for flexibility is an Internet standard. The Robustness Principle from RFC 1112 states "Be liberal in what you accept, and conservative in what you send." The realities of HTTP on the Internet make this advice mandatory.

[2] The line has been broken to allow it to fit the page; the original was one long line.

```
Method: GET
URL directory part: /
URL directory part: aya2003
URL directory part: /
URL directory part: daily
URL directory part: /
URL directory part: 20030715
URL directory part: /
URL directory part: thumbnails
URL directory part: /
image file
HTTP Version: HTTP/1.1
Key: Referer
Scheme: http
Valid hostname
URL directory part: aya2003
URL directory part: /
URL directory part: daily
URL directory part: /
URL directory part: 20030715
URL directory part: /
directory file
Key: Accept-Language
Accept language: en-us
Key: Accept-Encoding
Encoding: gzip
Encoding: deflate
Key: User-Agent
User agent: Mozilla
User agent version: 4.0
UA additional info: compatible
UA additional info: MSIE 5.5
UA additional info: Windows NT 5.0
UA additional info: (R1 1.3)
Key: Host
Valid hostname
Key: Cookie
Value: PHPSESSID=8ce5ac388fc6c1f07d213df818f5a2e9
```

Figure 4.2: The tokens resulting from parsing the HTTP request in Figure 4.1.

```
Referer: //search.msn.com/pass/results.asp?RS=CHECKED&FORM=
MSNH&v=1&q=&q=Geometry+Non+Euclidean&cp=1252
```

Note the missing `http:` in the URL for the referring web site. Another, example is:

```
Referer: www.av.com
```

The standard states that the `Referer` should be either an absolute or relative URI. This robot provides neither. These are but a small sampling of the nonstandard protocol implementations I encountered.

In addition to compliance with the standard, some web proxies modify header lines, presumably for privacy reasons. For example, the following header lines are some from the live data:[3]

```
Referer: XXXX:+++++++++++++++++++++++++++++++++++++++++++++
XXXXXXXXXXXXXXX:++++++++++++++
---------------: ----- -------
Proxy-~~~~~~~~~~: ~~~~~~~~~~
~~~~~~~: ~~~~:~~~~~~~~~~~~~~~~~~
```

For the second line above, note the missing standard-mandated space after the `:`. Some other proxies simply delete values, leaving a header line with a key only, such as `Referer:` or `Cookie:`. When dealing with these problems, the parser notes the incorrect or missing data (optionally by placing a token indicating the problem in the token stream) and continues.

4.3 Normal data

The normal data set is a collection of HTTP requests that were delivered to the University of New Mexico Computer Science departmental web server, as well as the web server

[3] The first example was truncated to allow it to fit the page.

hosting the domains aya.org, explorenm.com, and i-pi.com; these later data are referred to as the "pugillis" data after the name of the machine hosting the web sites.

All the data sets contain the entire HTTP request sent by the client to the server. Having the HTTP header lines allows testing for attacks not contained in the requested resource path (Section 2.3.2 presented the importance of this decision).

To collect the complete HTTP request delivered to the server, only a few options exist. A proxy could be placed between the web server and the network, and this proxy could collect the data. Because of concerns about performance by the UNM CS department, I did not use this approach. Instead, my data came from the second option of tapping the network by using an Ethernet hub to allow another computer to observe the packets destined for the web server. *snort* has the ability to extract the application-layer data from the individual network packets, saving one or more requests to a file[4]. Each data record includes the entire HTTP request sent by the client to the server, allowing the system to make use of the HTTP header lines and test for attacks that are not contained in the requested resource path.

snort collects the data with approximately a 99% accuracy rate. Problems arise from the requirement that the computer running *snort* must be fast enough to keep up with the network traffic; otherwise packets are lost, and the resulting stream is incomplete, and/or *snort* confuses the two directions of communication[5]. With the aid of custom scripts, I was able to repair or eliminate all of the *snort*-mangled requests. Cleaning the data required looking at thousands of unparsable HTTP requests to identify the problem, determine if it could be repaired, and if so, develop a method to automate the identification and repair or elimination of similarly damaged requests.

As shown in Chapter 2, previous researchers used data sets in which the attacks that occur in Internet-connected web server traffic have been filtered out. To show the effect of filtered and unfiltered data, I filtered attacks from the test data by running a combination of custom HTTP testing tools as well as using a set of rules for the signature-based IDS *snort*. Additionally, many questionable requests were manually inspected. This process

[4] Appendix A contains the configuration file, command line arguments, and scripts used for data collection.

[5] Performance problems with the computer collecting data at a local ISP prevented me from using data from their web server.

produced the data set that I refer to as the "filtered" data set. Although Code Red was removed[6], I saved the rest of the attacks from this data set as a separate data set that, when needed, can be added back into to the non-attack data. For the cs.unm.edu data, these attacks from the live data stream have been cleaned sufficiently to be used for training. I refer to the filtered data with these attack data added in as the "unfiltered" data set. I also created a data set consisting of the unfiltered data plus one of the Code Red attack variants inserted at a rate of 300 per hour; this rate corresponds to one reported rate [211], and is intermediate in the range of reported rates (from nine [81] to 12,000 [221] per hour) that occurred at the peak of the infection. I refer to this data as the "unfiltered + Code Red" data set. Some attacks from the pugillis data were discarded before their importance was realized, and others are not in a sufficiently clean state to be used. The training and testing phases could therefore be run on cs.unm.edu data sets with or without harmless attacks (i.e., attacks detected by *snort* but that were targeting programs or exploits for which the cs.unm.edu web server was not vulnerable).

The four web sites that were the source of the training and test data are:

aya.org This web site is for a non-profit organization of approximately 1,900 pilots. It was designed by a professional web site designer, and he used PHP and a MySQL database to provide dynamic content. In addition to the files on the web site, a large portion of the web site is generated from programs extracting data from the database.

cs.unm.edu This web site consists of departmental information as well as diverse, automatically-generated content provided by students, faculty, and staff. During the data collection time, the web server had 181,132 files on it.

explorenm.com This web site contains information about hiking and camping in New Mexico, and it is on the first page of Google for many web searches relating to its content. The web site uses Perl extensively and at the time the data was collected, it also made use of some PHP. Data for the site comes from a PostgreSQL database, and many web pages are generated from the data in the database in response to user actions such as where they click on the map of New Mexico.

i-pi.com This web site is a simple one composed of files and little dynamic content. The content drawing the most attention is the cave biology research of Diana Northup, a

[6] I deleted the Code Red attacks before I realized they would be useful.

Web site	Data set ID	Filtered	Unfiltered
aya.org	2005-01-25	84,241	85,666
	2005-01-29	19,962	20,266
	2005-02-05	39,468	39,506
	2005-02-12	45,785	46,353
cs.unm.edu	2004-11-12	390,911	450,002
	2004-11-17	458,375	570,165
	2004-12-06	468,009	475,393
	2004-12-18	440,444	441,476
	2005-01-01	404,901	409,490
	2005-01-22	238,953	243,813
explorenm.com	2005-01-25	74,755	76,046
	2005-01-29	17,262	17,438
	2005-02-05	37,994	38,043
	2005-02-12	37,691	38,537
i-pi.com	2005-01-25	14,123	14,616
	2005-01-29	3,308	3,312
	2005-02-05	9,700	9,702
	2005-02-12	10,441	10,597

Table 4.1: The web server data set sizes (in number of requests). The difference between the filtered and unfiltered data reflects the number of harmless attacks in requests to the web server.

professor emerita of the UNM general libraries and a visiting associate professor in the Biology department at UNM.

The test data sets are approximately one week of data each. The actual size of the collection was limited on FreeBSD by a limit of 32,766 files in a single directory. Therefore, the sets are not all exactly seven days long, but they are comparably-sized. Within a data set, the requests are not temporally ordered, but between data sets, temporal order is preserved; Section 5.1 shows this has little effect on the DFA learning. Some weeks were lost due to technical difficulties, explaining gaps longer than a week in the data set dates. Table 4.1 summarizes the data sets and their sizes. For the cs.unm.edu 2004-11-12 data set, Table 4.2 provides some statistics on the request lengths in the training data.

Measure	**Filtered**		**Unfiltered**		**Unfiltered + Code Red**	
	Character	**Token**	**Character**	**Token**	**Character**	**Token**
Maximum	3,894	841	106,064	841	106,064	841
Minimum	14	4	3	4	3	4
Mean	343	41	464	40	569	32
Mode	319	33	319	32	1,411	14
Stddev	126	16	2,446	17	575	18

Table 4.2: Statistics about request lengths measured as number of characters and number of tokens. These data are calculated from the requests in the cs.unm.edu 2004-11-12 data.

4.4 Attack data

My attack database contains 63 attacks, some of which are different examples of the same vulnerability—either a different exploit for the same vulnerability or an exploit for the vulnerability on a different operating system. The reason for including the variants of attacks is that they are all different, some notably so. Some will be easier to detect than others for some of the algorithms. As one example, some of the Nimda attacks are short enough to be detected by the length algorithm, while others are closer to an average length.

I collected the attacks from the following sources: Attacks against web servers I was testing (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 I had to modify the program before it would produce malicious web requests. Note that I did not test to verify whether the attacks produced could actually compromise the targeted web application.

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.

As a comparison to the normal data sets collected from live web servers, the same statistics for the attack data are presented in Table 4.3.

My test data set is more extensive than those used in most previous studies. Researchers using the 1999 Lincoln Labs data set (Section 2.3.2) have only four web attacks. Researchers realizing this limitation obtained their own data. Kruegel and Vigna [146] used 11 attacks, a portion of which were chosen specifically because they targeted soft-

Measure	Character	Token
Maximum	54,494	8180
Minimum	9	4
Mean	2922	158
Mode	91	19
Stddev	10,353	954

Table 4.3: Statistics about request lengths measured as number of characters and number of tokens. These data are calculated from the requests in the attack database.

ware from some of the web servers from which they obtained their training data. Tombini et al. [248] mentioned using 56 attacks in part of their testing, but the number of attacks they used varied with the test performed. For the results in this dissertation, every algorithm was tested against every attack not in its training set—e.g., if Code Red was in the training set, it was not in the attack data set.

The attack database for these tests is at `http://www.i-pi.com/HTTP-attacks-JoCN-2006`.

4.5 Summary

The test framework I developed makes it possible to test algorithms in a consistent manner. My testing is more rigorous than most IDS testing reported in the literature because I tested the algorithms under identical circumstances on identical data.

Parsing HTTP is straightforward, although the collection of clients that incorrectly implement the standard is substantial. A parser must be able to handle common deviations from the standard.

The data I collected is representative of requests delivered to live web servers, and is sufficient for the testing described in the following chapters. The attack data are more representative of the diversity of attacks than most data sets used previously.

Chapter 5

Experiments and results

Chapters 1 and 2 both discussed the weak testing common for IDSs. One of the contributions of my research addresses this deficiency, and this chapter presents results of several types of testing. Section 5.1 provides support for my previous claims that different web sites receive different requests, HTTP is a nonstationary data source, and attack diversity is higher than normal request diversity. The second type of testing investigates the rate of learning of the DFA and n-grams, the two algorithms new to HTTP. These results are in Section 5.2. The space required by an algorithm as it runs affects its suitability for production use; Section 5.3 provides this information for the algorithms. Information about the algorithm accuracy is presented in Section 5.4. These results provide guidance about which algorithms are suitable for HTTP and how to proceed to improve HTTP anomaly detection beyond this dissertation. An important part of the accuracy for the DFA and n-gram algorithms is the heuristics described in Section 3.10. Support for the heuristic effectiveness is in Section 5.5. Finally, to show that the results are not an artifact of my tuning the heuristics to a specific data set, Section 5.6 shows similar accuracy results on a pugillis data set I had not inspected until after the Ph.D. defense.

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) [103]. In the ROC curve plots in this chapter, true or false positives are represented 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. 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; a few required a different scale to present the data, and this fact is noted where applicable.

McHugh noted several potential problems presenting IDS test results with ROC curves [186]. His first objection was that some researchers presented results using the ROC curve, and this curve consisted one point. The researchers made a complete curve by assuming that the curve passes through (0,0) and (1,1). I ran tests in a manner that allowed me to produce ROC curves containing 127 points showing the effects of different thresholds. I do not assume that the curve goes through any points, and 127 points are sufficient to produce a good curve. 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 dissertation, this unit of analysis is always one HTTP request.

Gu et al. [100] recently proposed a single measure as an alternative to ROC curves for measuring IDS effectiveness. Future work includes comparing their measure to the traditional ROC curve.

5.1 Support for claims about HTTP

Chapter 2 contained several several claims about HTTP request data. Section 2.4.3 claimed that different web sites receive different requests. This diversity between web sites might be exploited by an IDS to force an attacker to customize her attack to the web site, eliminating the “write once, run everywhere” attack. Section 5.1.1 shows that some algorithms can detect this diversity, while others cannot.

Section 2.4.5 claimed that HTTP is a nonstationary data source, and that this property is difficult for many anomaly detection algorithms. Because web servers change over time, the accuracy of an algorithm trained on one data set might degrade over time unless the algorithm’s model changes with the protected web site. Section 5.1.2 shows this effect on

Train on:	**cs.unm.edu**		**aya.org**		**i-pi.com**		**explorenm.com**	
FP	**frac**	**/day**	**frac**	**/day**	**frac**	**/day**	**frac**	**/day**
Test on:								
cs.unm.edu	0.005	484	0.101	9,267	0.066	6,053	0.073	6,666
aya.org	0.031	154	0.005	27	0.006	29	0.020	101
i-pi.com	0.109	90	0.067	56	0.003	2	0.084	69
explorenm.com	0.118	507	0.074	321	0.014	61	0.014	61

Table 5.1: DFA accuracy across web sites: The DFA induced from each web site is tested against the other web sites. The units are first false positive fraction, then false positives per day. The results are for a continually learning DFA with a true positive rate of 0.852.

several of the HTTP IDS algorithms.

Section 2.4.4 claimed that attack diversity is higher than normal request diversity. Section 5.1.3 supports this claim by showing that attacks against web servers and web applications are more diverse than non-attack traffic.

5.1.1 Web sites are diverse

Section 2.4.3 claimed that different web sites receive different profiles of requests. Testing an algorithm trained on one web site on data from other web sites shows if the algorithm is able to distinguish between requests delivered to different web sites. For all of these tests, I assumed that the required accuracy was a relatively generous true positive fraction of 0.852. This choice allowed more algorithms into the comparison.

Table 5.1 shows the results of testing the DFA induced from filtered data for each web server on the other web sites, while Table 5.2 and 5.3 present the same results for 6-grams and the Mahalanobis distance character distribution respectively. The linear combination, length, and χ^2 of idealized character distribution were never able to reach a true positive rate of 0.852, hence they have no corresponding tables. The data also show that the pugillis sites are similar between themselves, but further separated from the cs.unm.edu data. For example, the DFA trained on i-pi.com data is not a bad fit for the aya.org data when viewed as false positive fraction. However, the false positives per day are higher; aya.org receives more traffic than i-pi.com.

Train on:	**cs.unm.edu**		**aya.org**		**i-pi.com**		**explorenm.com**	
FP	**frac**	**/day**	**frac**	**/day**	**frac**	**/day**	**frac**	**/day**
Test on:								
cs.unm.edu	0.035	3,228	0.162	14,865	0.061	5,578	0.494	45,244
aya.org	0.513	2,562	0.001	6	0.001	6	0.374	1,869
i-pi.com	0.686	568	0.454	375	0.000	0	0.562	465
explorenm.com	0.423	1,825	0.132	569	0.002	8	0.002	8

Table 5.2: 6-gram accuracy across web sites: The 6-gram induced from each web site is tested against the other web sites. The units are first false positive fraction, then false positives per day. The results are for a a true positive rate of 0.852.

Train on:	**cs.unm.edu**		**aya.org**		**i-pi.com**		**explorenm.com**	
FP	**frac**	**/day**	**frac**	**/day**	**frac**	**/day**	**frac**	**/day**
Test on:								
cs.unm.edu	0.874	80,121	1.000	91,675	0.999	91,620	1.000	91,673
aya.org	0.982	4,899	0.060	299	1.000	4,306	1.000	4,990
i-pi.com	0.734	607	1.000	827	0.151	125	1.000	827
explorenm.com	0.895	3,862	1.000	4,315	0.998	4,306	0.096	413

Table 5.3: Mahalanobis distance between character distributions accuracy across web sites: The reference distribution induced from each web site is tested against the other web sites. The units are first false positive fraction, then false positives per day. The results are for a a true positive rate of 0.852.

The data in Table 5.1 show that the DFA can distinguish between web sites; in all cases, the smallest fraction of false positives for a fixed true positive rate is for the site where it was trained. The 6-gram results, Table 5.2, show a similar pattern. The Mahalanobis distance results in Table 5.3 show that this measure does not always distinguish between web sites. For example, when trained on the cs.unm.edu data, the best results are with the i-pi.com data. However, for the aya.org, i-pi.com, and explorenm.com, this measure is substantially more accurate on the site for which it was trained.

5.1.2 HTTP is a nonstationary data source

Three of the algorithms (DFA, n-grams, and Mahalanobis distance) have well-defined and implemented methods for nonstationary data. The χ^2 distance and enumerated or random

Algorithm	**Data set**	**FP fraction**	**FP fraction**
		Static	**Adaptive**
DFA	2004-11-17	0.076	0.032
DFA	2005-01-22	0.246	0.017
6-grams	2004-11-17	0.168	0.165
6-grams	2005-01-22	0.318	0.268
Mahalanobis distance	2004-11-17	0.997	0.997
Mahalanobis distance	2005-01-22	0.991	0.991

Table 5.4: Effect of continuous learning on false positives: False positives are shown for the algorithms trained on the cs.unm.edu 2004-11-12 data and applied to the CS data from 2004-11-17 and 2005-01-22. False positive (FP) fractions are reported for a true positive fraction of 0.852. The algorithm labeled *static* learns only once and does not update its model as it runs, whereas the algorithm labeled *adaptive* does.

parameters algorithms have no provision for updating as they run, so when the web site changes enough, re-training will be required. This limitation therefore applies to the linear combination as well. Table 5.4 shows the difference in accuracy resulting from enabling the portion of the algorithm that handles nonstationary data. For the adaptive version of the algorithms, they were tested on all of the intermediate data sets[1], allowing them to adapt as the web site changed.

A DFA that continues to learn is notably more accurate than one that does not. Even over approximately one week's data, the static algorithm had a false-positive rate of 7.6%, while the adaptive version rate was 3.2%. After approximately two months, the static version was miscategorizing nearly 25% of the requests, compared with 1.7% for the adaptive version. The fact that the false positive rate for the adaptive version dropped over the test interval shows that the adaptive DFA continues to develop a better model of the web site. Tracking the web site as it changes is therefore a requirement for this algorithm. For 6-grams, the adaptive algorithm improves accuracy, but the accuracy slowly degrades over time.

For the Mahalanobis distance, enabling the ability to learn from normal produced no change. The test object only adds new instances when they are similar enough to be able to believe that the request is not an attack. The default value for this similarity is 0.9 (Section 3.10). However, none of the test data reached this similarity threshold, so no new

[1] These intermediate results are not reported to save space

Measure	**Attacks**	**cs.unm.edu**	**aya.org**	**i-pi.com**	**explorenm.com**
Minimum	0.00	0.01	0.02	0.02	0.03
Maximum	1.00	0.98	1.00	1.00	0.99
Mean	0.15	0.27	0.33	0.30	0.32
Median	0.08	0.19	0.27	0.27	0.29
Standard deviation	0.20	0.19	0.21	0.18	0.17

Table 5.5: Inter-request diversity for attacks and within a data set.

instances were added to the model.

5.1.3 Attack diversity

Section 2.4.4 claimed that attack diversity is higher than normal request diversity. As a quantitative test, I used the DFA induction algorithm to generate a DFA corresponding to each attack, and then compared all of the other attacks to this DFA. For a non-attack comparison, using the OpenSSL pseudo-random number generator seeded from `/dev/urandom`, I selected 100 samples of 100 random requests from the four web sites, using the 2004-11-12 cs.unm.edu data and the 2005-01-25 pugillis data. These results were combined by averaging the values. A summary of these results are in Table 5.5.

Note that the attack diversity is higher, as indicated by the lower similarity mean and median values. The two attacks that were considered the same were two related attacks that, when the default heuristics were applied, resulted in the same list of tokens.

5.2 DFA and n-gram learning

The DFA and n-gram algorithms are new to HTTP. This section shows results relating to the learning these algorithms perform. Section 5.2.1 shows the training data are sufficient for the algorithms. Section 5.2.2 shows that the order of the training instances makes little difference for the accuracy of the resulting DFA.

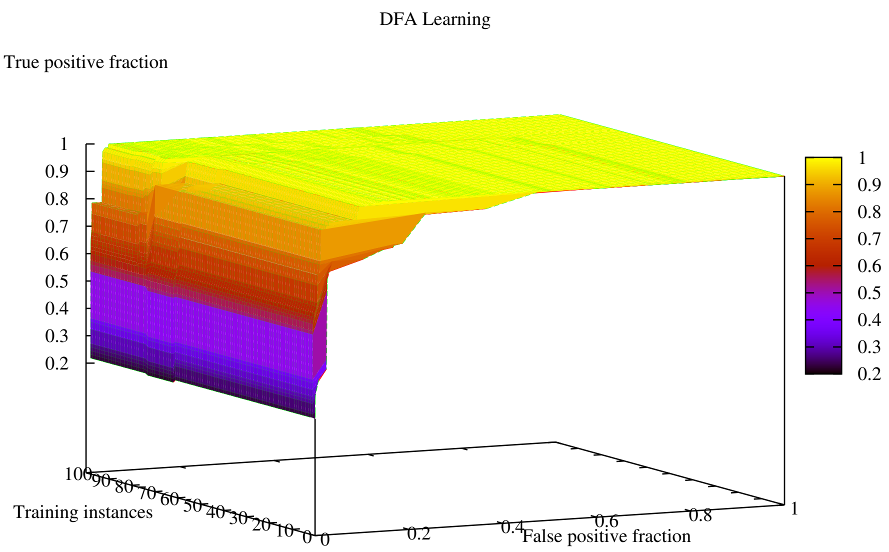


Figure 5.1: DFA learning for the first 100 training instances.

5.2.1 Sufficient data for learning

In order to show that my training data is sufficient for inducing a DFA or a set of n-grams, I saved the state after each training instance from the cs.unm.edu 2004-11-12 filtered data. I then used this saved state to test against the cs.unm.edu 2004-11-17 filtered data. The results for the first 100 training instances are in Figure 5.1 for the DFA. Note that the ROC curve at the back of the plot (after 100 training instances have been processed) is close to the final shape (shown in Figure 5.11) by around 50 training instances—the full training data set contains 390,911 requests (Table 4.1).

Figure 5.2 for 6-grams. In this plot, a ROC curve is shown for every 1000 training instances. More training data is required for this algorithm to become accurate, but the ROC curve approximates the final curve after processing 40,000 training instances.

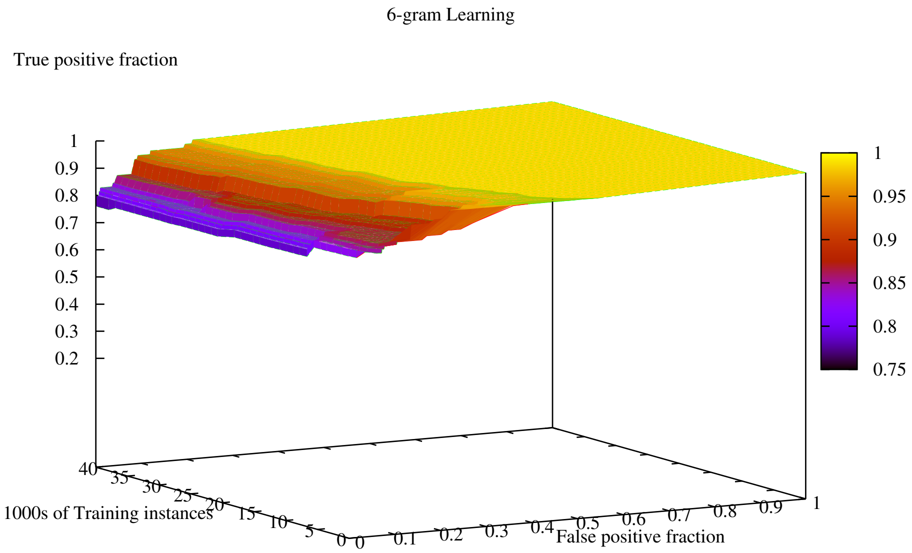


Figure 5.2: 6-gram learning for the first 100 training instances.

5.2.2 DFA and order of training data

Within a data set, the instances are not temporally ordered (Section 4.3). During DFA learning, the state merging implies that the order of the training instances might affect the results. To show this is not the case, I tried different orderings of the filtered cs.unm.edu 2004-11-12 data to train the DFA: sorted by client IP address (forward, the default order), the reverse of this order, and three random orderings, where the random numbers were obtained using the code described in Section 5.1.3. Figure 5.3 shows that all of the resulting DFAs perform similarly.

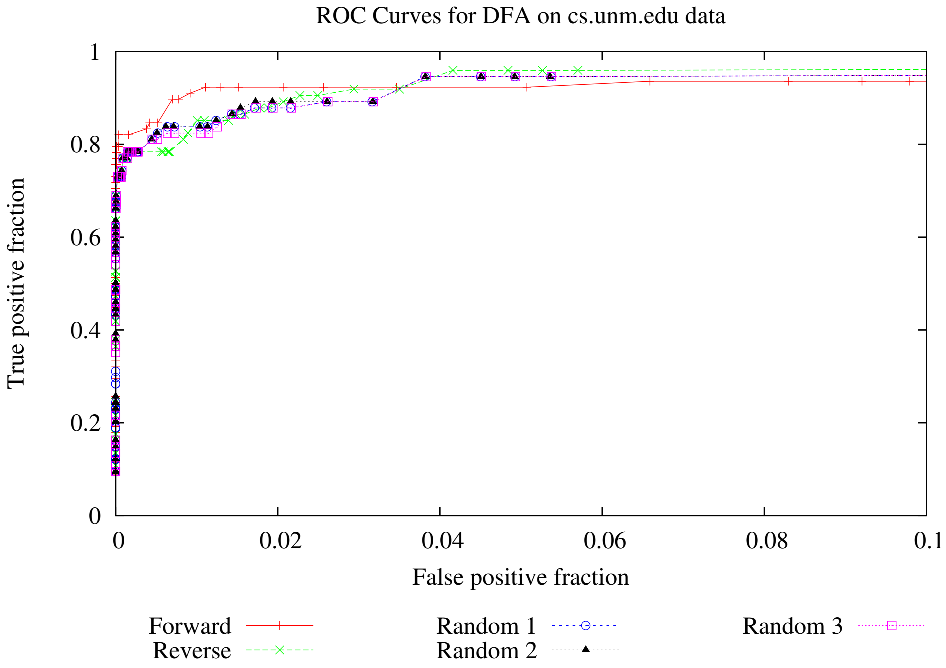


Figure 5.3: ROC curves for DFAs trained on different orderings of the filtered CS 2004-11-12 data.

5.3 Space requirements of the algorithms

Criterion 3 in Section 1.2 stated that an IDS should not place an undue burden on the machine it protects. One measure of the burden is the amount of memory required for the algorithm's model of normal. The size of the saved state is representative of the space requirement of the algorithm (as I implemented it). Table 5.6 presents the storage needs for each of the algorithms. Compare keeping all the 6-grams in memory, using 77–82MB to the DFA using 4.4MB. The Markov model's extra space requirements are clearly not a worthwhile trade-off, because as Section 5.4 presents, its false positive rate is so much worse than other algorithms. The other statistical measures require little space.

In writing the algorithms, I stressed speed and accuracy of coding, and I did not attempt to optimize the storage. However, the order of magnitude of storage requirements

Algorithm	**Filtered**	**Unfiltered**	**Unfiltered + Code Red**
Mahalanobis distance	7,944	8,971	8,930
χ^2 of idealized character distribution	3,366	3,901	3,849
Length	139	145	146
3-grams	17,090,308	17,747,108	17,740,446
4-grams	31,801,835	33,236,926	33,227,178
5-grams	52,086,245	55,107,800	55,095,067
6-grams	76,950,157	81,956,324	81,940,620
7-grams	106,566,643	114,275,087	114,256,549
DFA	4,435,615	4,668,452	4,666,383
Markov model	7,849,033	8,224,376	8,220,620
Linear combination	919,654	986,661	986,663

Table 5.6: The size in bytes of the saved state for the algorithms when trained on the 2004-11-12 cs.unm.edu data.

Data set	**Nodes**	**Edges**
Filtered	7,843	61,585
Unfiltered	7,856	62,941
Unfiltered + Code Red	7,855	62,911

Table 5.7: The size of the induced DFA in terms of nodes and edges for each of the training data sets.

is unlikely to change with more efficient implementations; the information required by the algorithm to function must be stored. Compression algorithms could reduce the storage space with a corresponding increase in CPU time and memory. For the n-grams, stide used a forest of trees, and this approach would be an example of a more efficient storage system [116].

The n-grams and DFA data structures have size metrics other than number of bytes. Figure 5.4 shows the growth rate in number of distinct n-grams and bytes for each of the test data sets. As n gets larger, the number of distinct n-grams grows at a rate slightly worse than linear, notably better than the worst case of exponential growth. Table 5.7 presents the DFA size in terms of nodes and edges for each of the training data sets. A good sign for potential production deployment is the small size of the DFA relative to the size of the training data (data statistics including size were presented in Section 4.3).

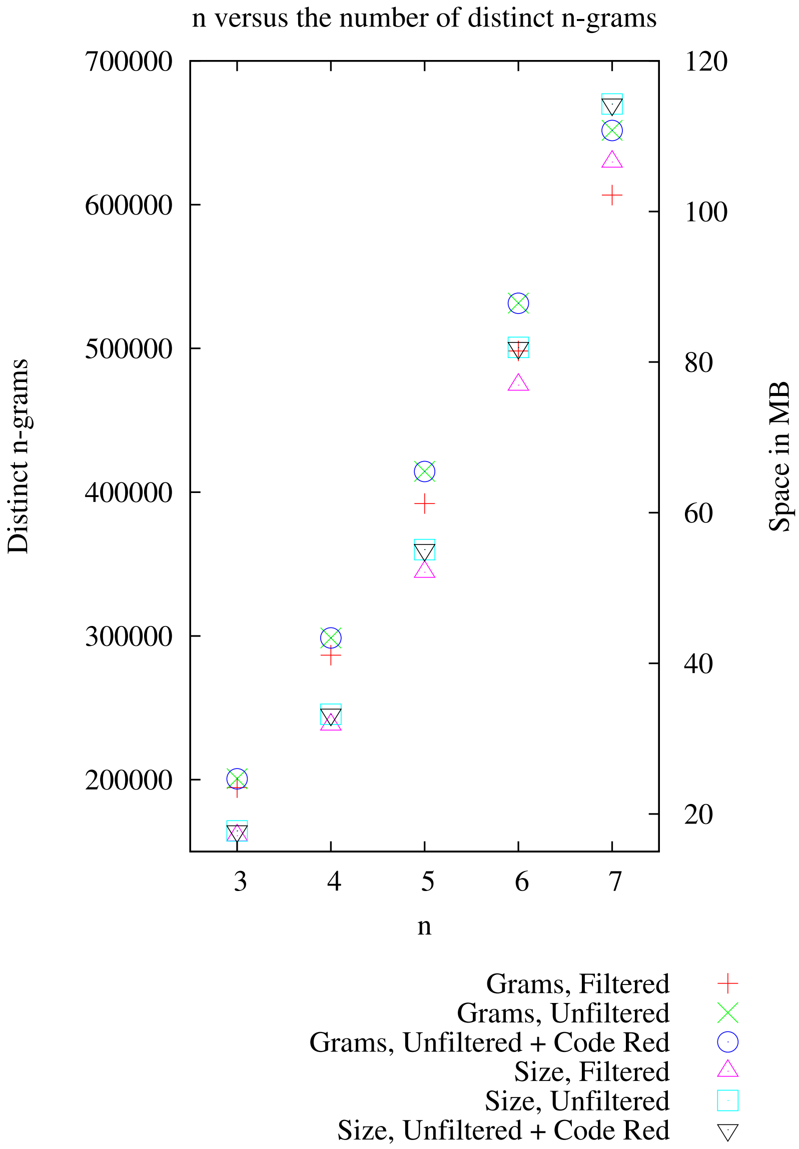


Figure 5.4: The growth in bytes and number of distinct n-grams versus n for the cs.unm.edu 2004-11-12 data set.

5.4 Algorithm accuracy

The primary reason for using an anomaly detection system is to detect anomalies. The tests I performed used the data described in Sections 4.3 and 4.4, and each algorithm was tested under identical circumstances on the same data. In all cases, the test unit is a single HTTP request, regardless of whether it was presented as a character string or as a list of tokens. When training on unfiltered data, the attacks in the training data were omitted from the attack test data set.

5.4.1 Length

Figure 5.5 shows the results of testing the length algorithm. Accuracy, which was below 80% true positive at best, drops substantially when the algorithm is trained on data containing attacks. The results of the test on pugillis data show this algorithm is even less accurate on this data.

5.4.2 Mahalanobis distance

The results for Mahalanobis distance are in Figure 5.6. For this algorithm on this data, using the unfiltered data results in more accuracy than the filtered, but when Code Red is added into the training data, the accuracy drops to below that of the filtered data. In none of these versions is the accuracy even close to sufficient for production use. For this algorithm, the pugillis data represents a notably easier test data set than the cs.unm.edu data.

5.4.3 χ^2 distance

Figure 5.7 contains the χ^2 distance results. This algorithm performs poorly on all data sets, with a true positive rate at or below 50%.

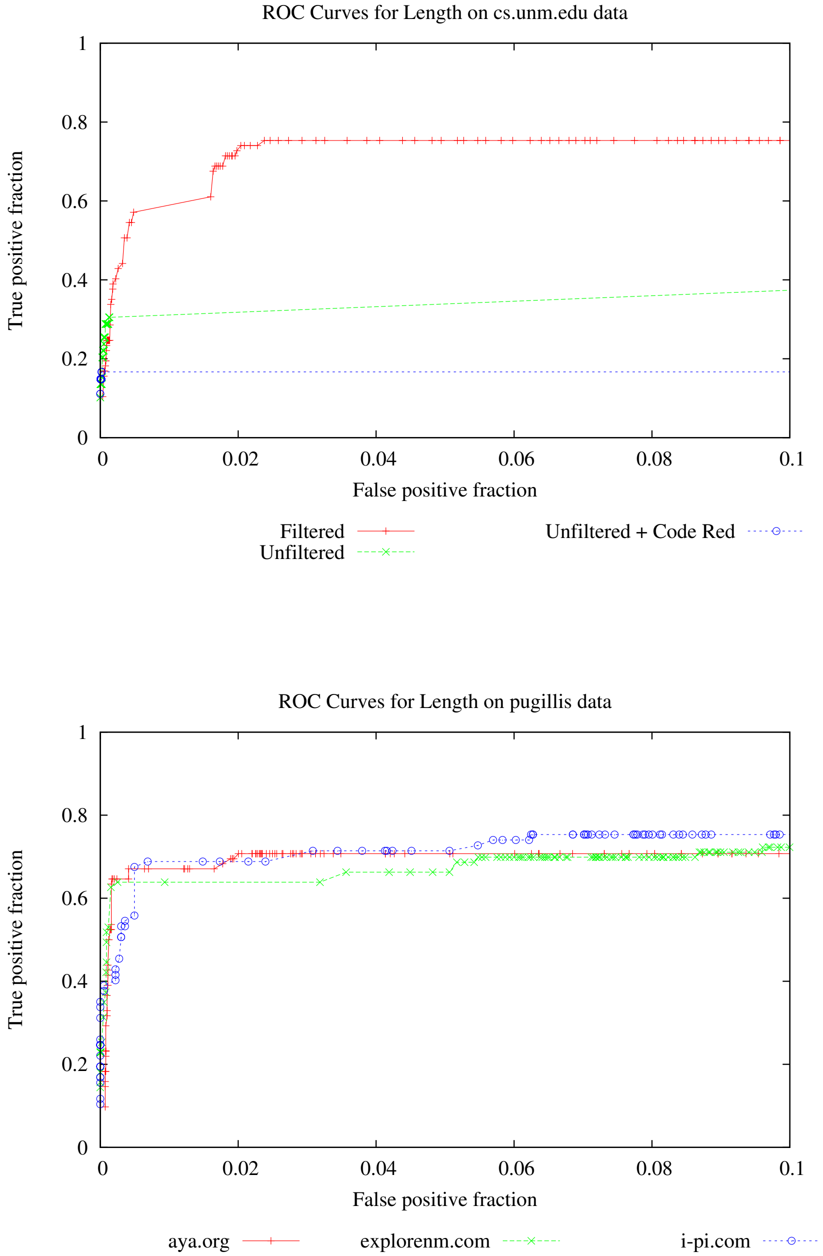
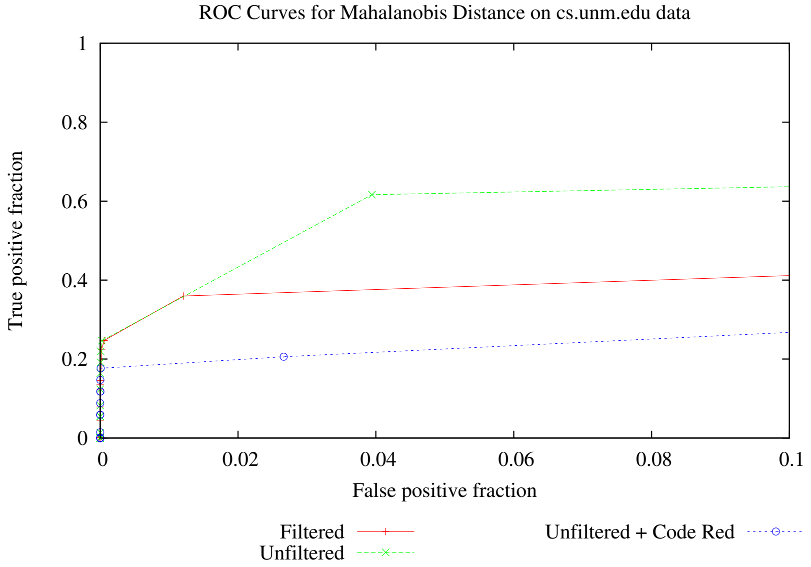


Figure 5.5: Receiver Operating Characteristic curves showing the accuracy of the length algorithm.



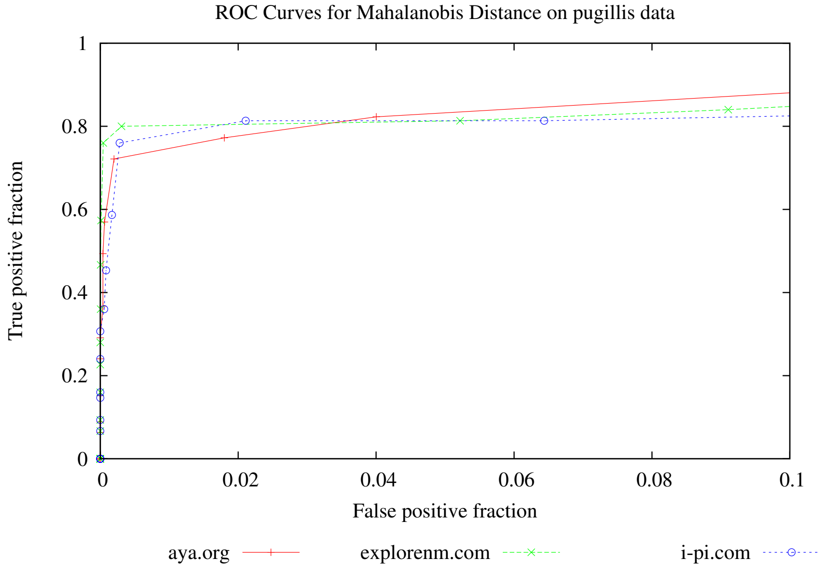
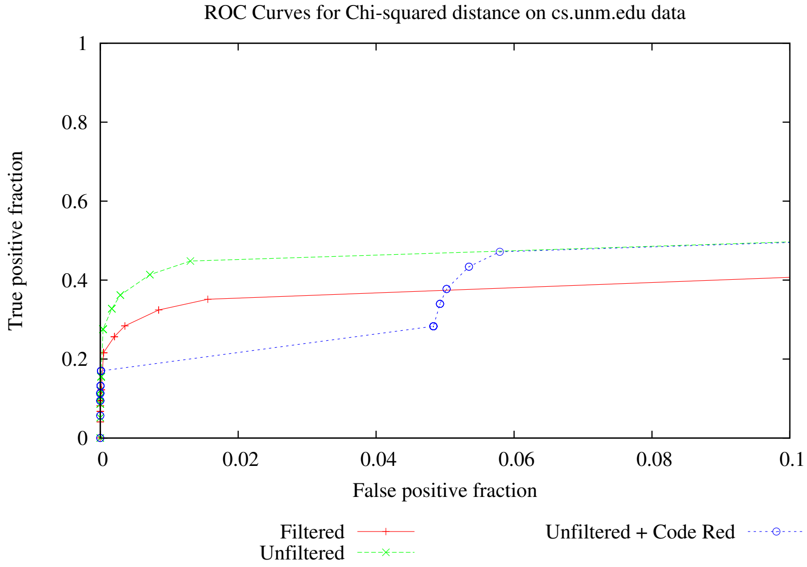


Figure 5.6: Receiver Operating Characteristic curves showing the accuracy of the Mahalanobis distance algorithm.



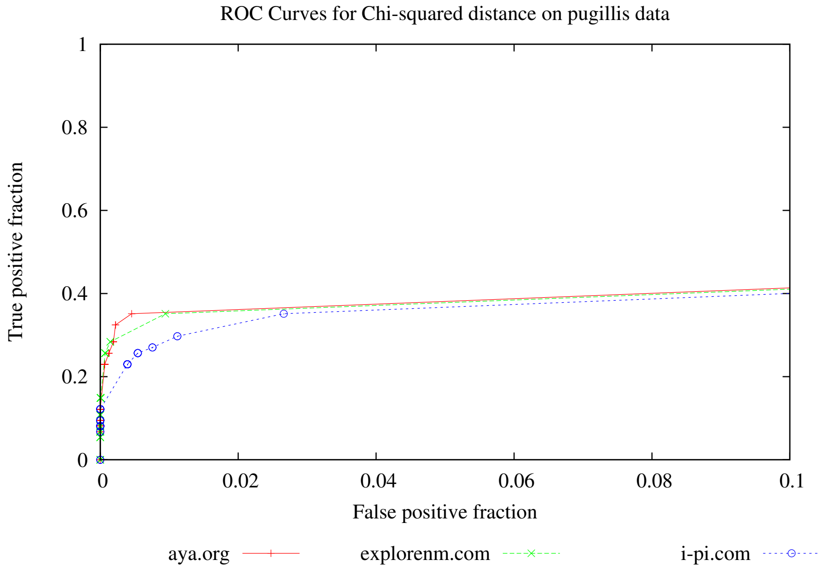


Figure 5.7: Receiver Operating Characteristic curves showing the accuracy of the χ^2 distance algorithm.

5.4.4 Markov model

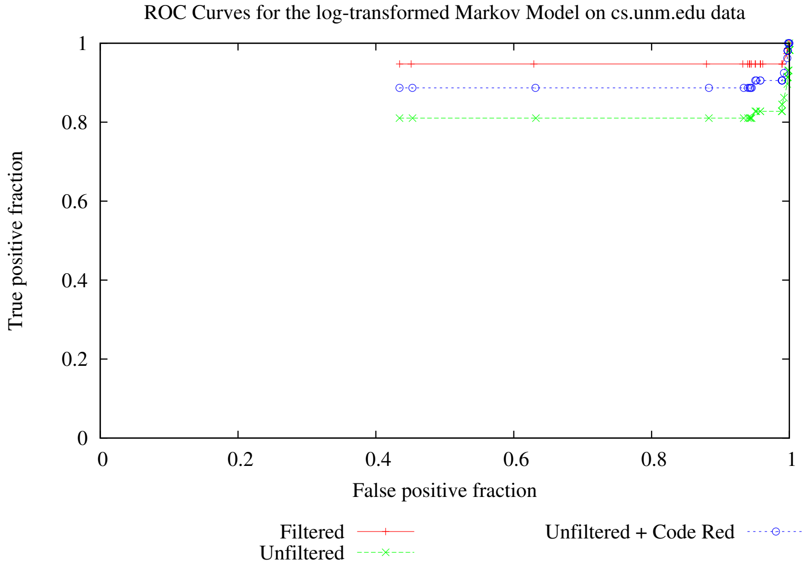
The Markov model result values are in $[0, 10^{-13}]$ with many values as small as 10^{-300}. These small values make it appear that the algorithm identifies everything (both normal traffic and attacks) as abnormal. To better understand these results, Figure 5.8 shows the data plot where the similarity 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 chapter). The log transformed Markov model provides 94% accuracy on filtered cs.unm.edu data, but with an unacceptable false positive rate. The results on the pugillis data show an even better true positive rate. Again, the false positive rate, while better than on the cs.unm.edu data, is unacceptable.

5.4.5 Linear combination

The linear combination results are in Figure 5.9. When tested on the cs.unm.edu data, it is notably more accurate on filtered data, but the true positive rate is only around 60%. The pugillis data shows even less accuracy. Neither is accurate enough to consider for production use.

5.4.6 n-grams

Earlier testing, not included in this dissertation, showed that $n = 6$ was an optimal value when considering the mix of accuracy and data set size; $n = 7$ was slightly more accurate, but was large enough to exhaust the system memory on some tests, even with the heuristics. The results for 6-grams are in Figure 5.10. In spite of 6-grams ability to distinguish between regions of a request, it was more accurate on cs.unm.edu filtered data than either unfiltered data set. The algorithm is more accurate on the pugillis data than it was on the cs.unm.edu data.



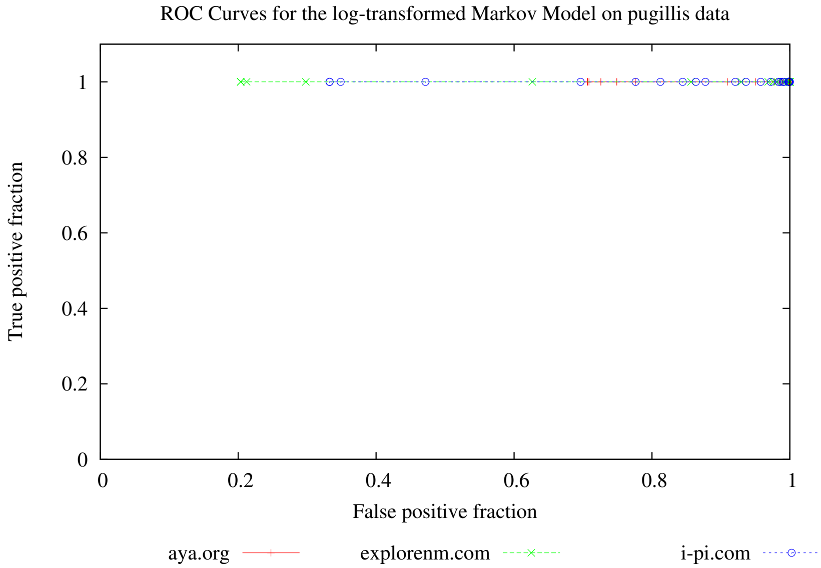


Figure 5.8: Receiver Operating Characteristic curves showing the accuracy of the Markov model algorithm.

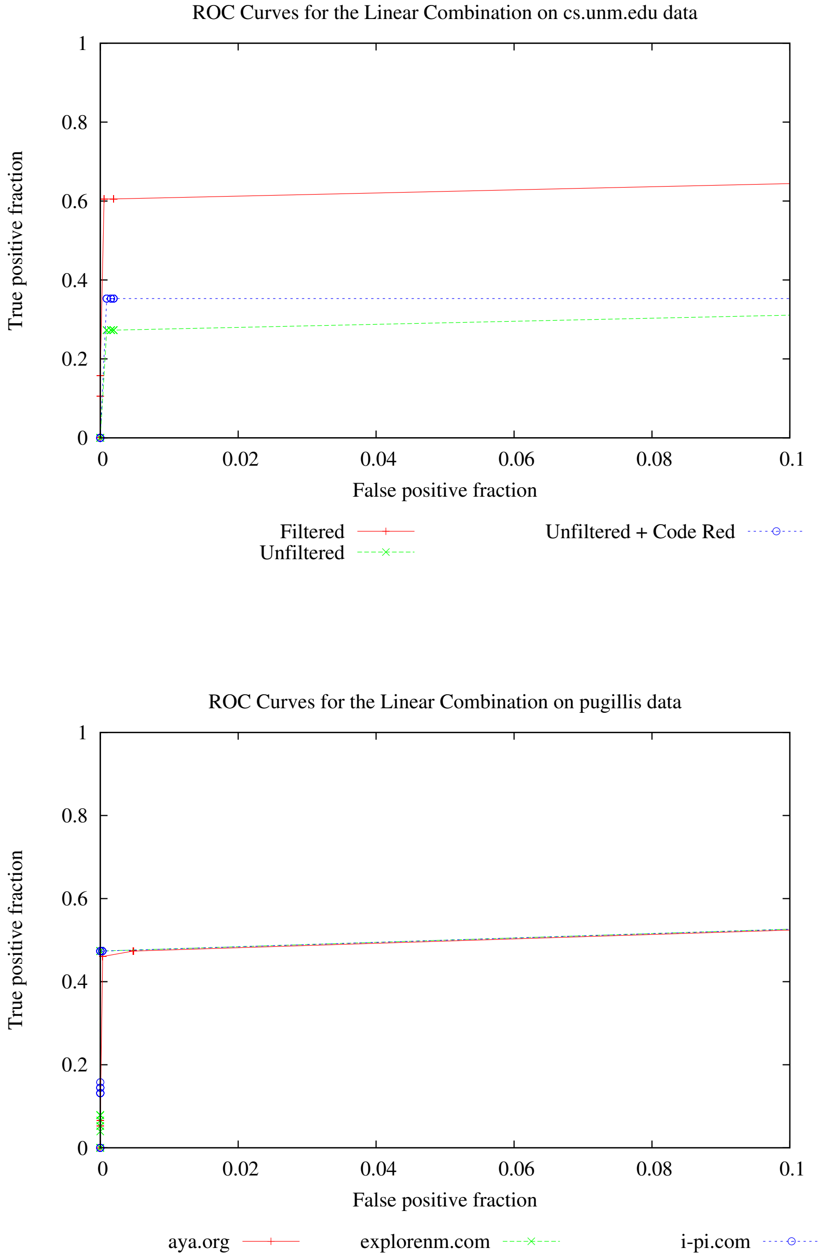
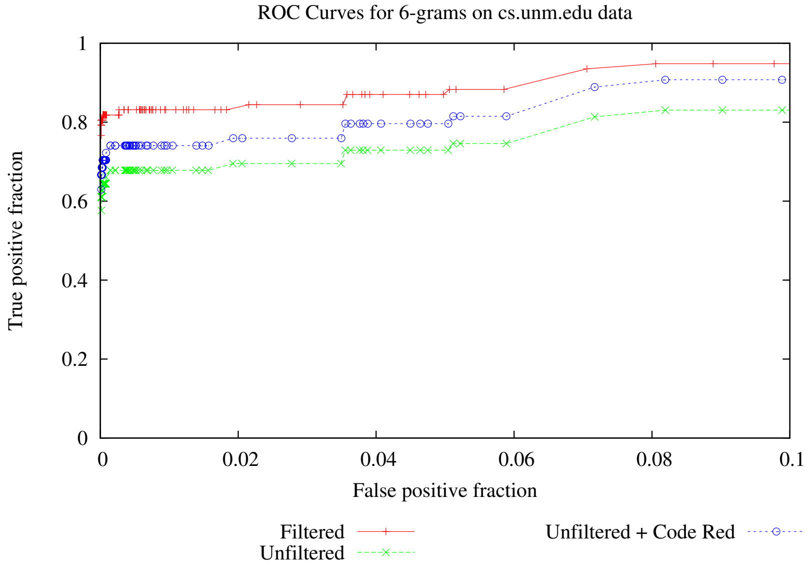


Figure 5.9: Receiver Operating Characteristic curves showing the accuracy of the linear combination algorithm.



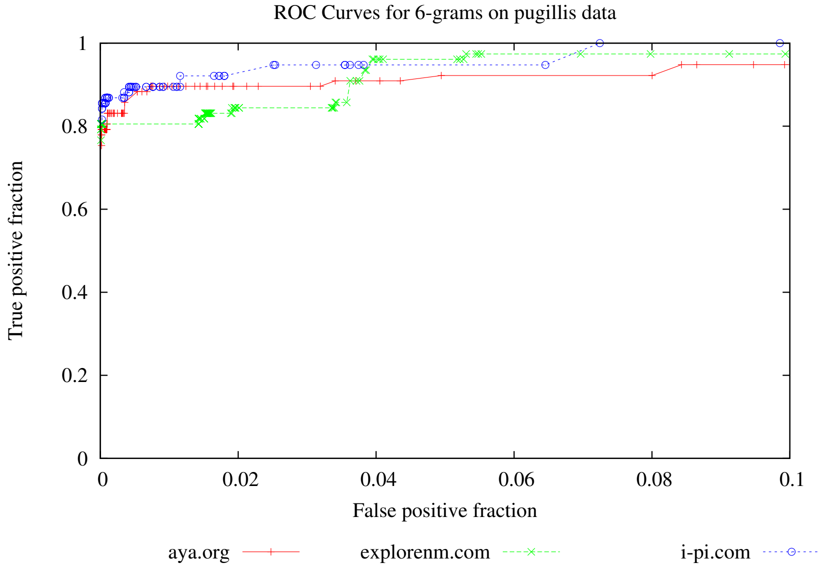
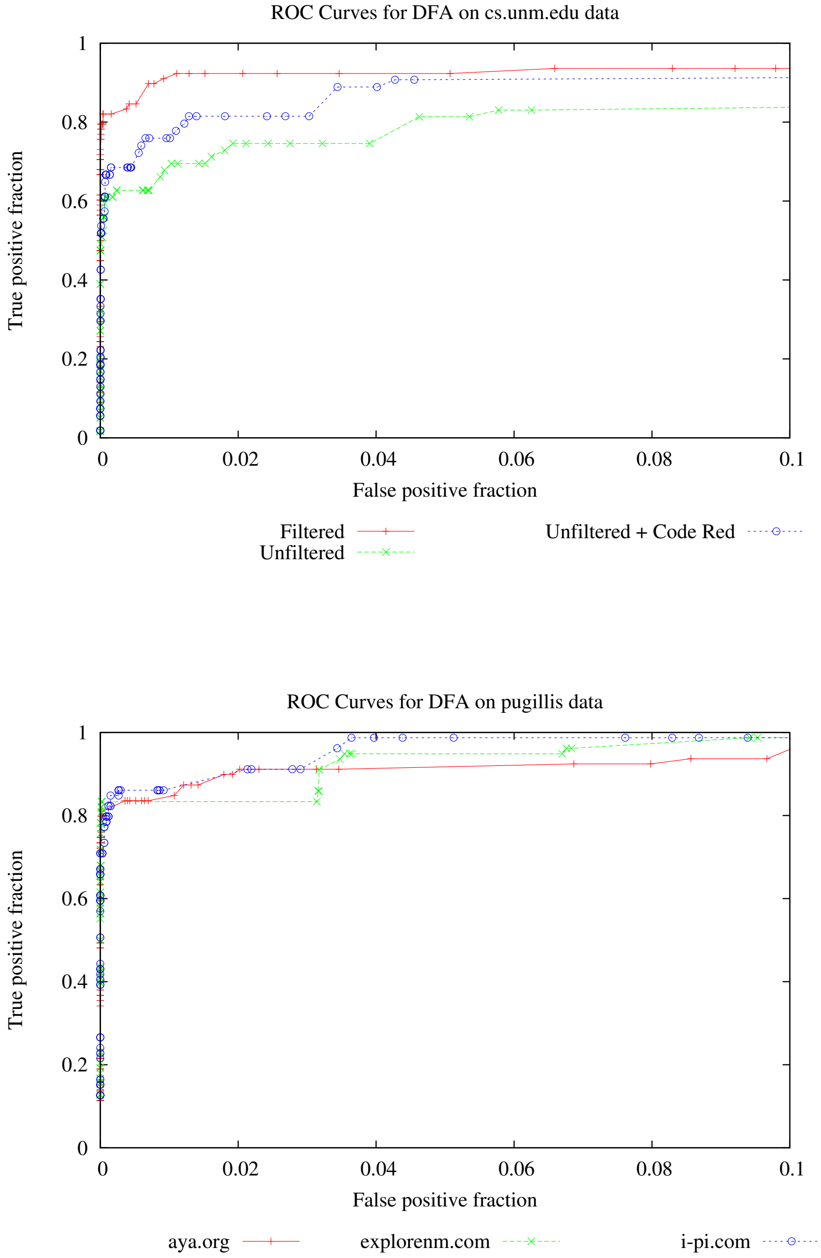


Figure 5.10: Receiver Operating Characteristic curves showing the accuracy of the 6-gram algorithm.

5.4.7 DFA

Figure 5.11 shows the DFA accuracy. At lower false positive rates, the true positive rate is best on the cs.unm.edu filtered data. The pugillis data all show nearly the same 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%.



ROC Curves for DFA on cs.unm.edu data

ROC Curves for DFA on pugillis data

Figure 5.11: Receiver Operating Characteristic curves showing the accuracy of the DFA algorithm.

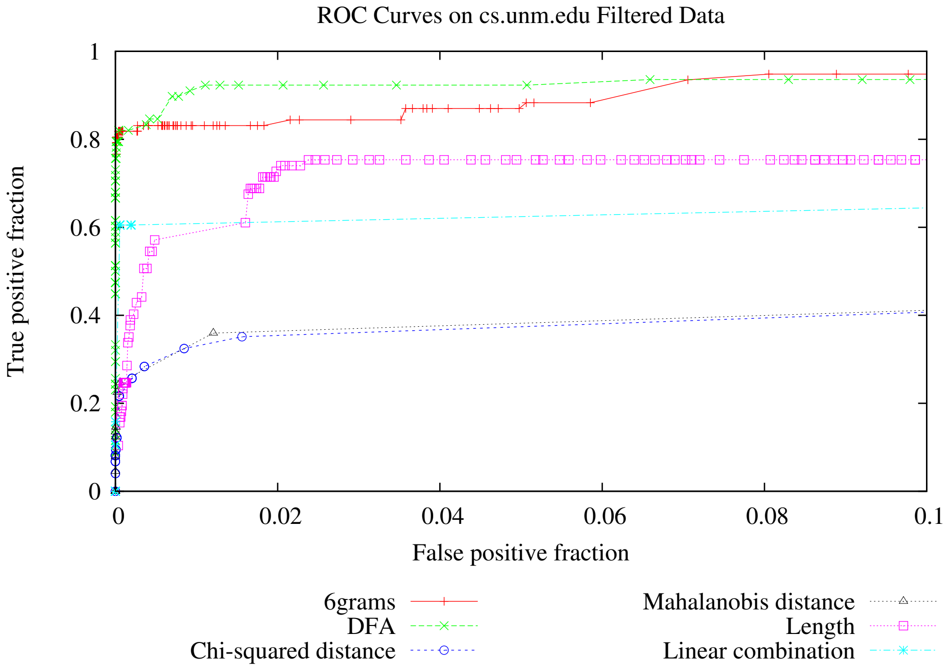


Figure 5.12: Receiver Operating Characteristic curves showing the relative accuracy of the DFA, 6-grams, χ^2 distance between the idealized character distribution and test distribution, and the length algorithms when tested on cs.unm.edu filtered data.

5.4.8 Algorithm comparison

To better compare the algorithms, Figures 5.12 shows on one plot the DFA, 6-grams, χ^2 distance, Mahalanobis distance, length, and the linear combination algorithms using the filtered cs.unm.edu data. The DFA and 6-grams have notably better true and false positive rates than the other algorithms. Figures 5.13, 5.14, and 5.15 show similar results for aya.org, i-pi.com, and explorenm.com respectively, although the Mahalanobis distance showed comparable accuracy on the i-pi.com data. On the explorenm.com data, it achieved a similar true positive rate, but only at a much larger false positive rate.

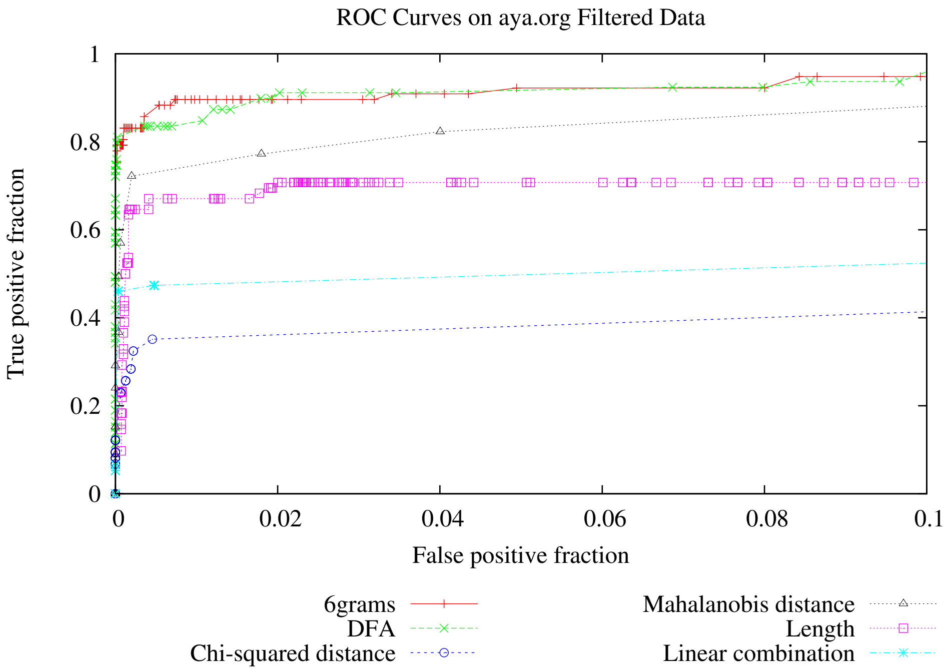


Figure 5.13: Receiver Operating Characteristic curves showing the relative accuracy of the DFA, 6-grams, χ^2 distance between the idealized character distribution and test distribution, Mahalanobis distance between character distributions and the length algorithms when tested on aya.org data.

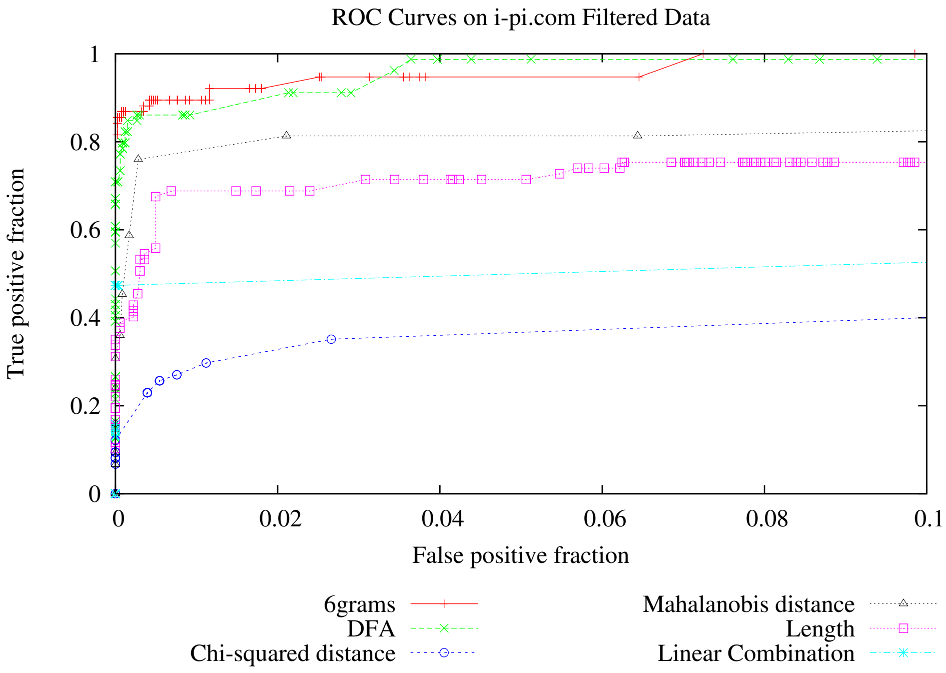


Figure 5.14: Receiver Operating Characteristic curves showing the relative accuracy of the DFA, 6-grams, χ^2 distance between the idealized character distribution and test distribution, Mahalanobis distance between character distributions and the length algorithms when tested on i-pi.com data.

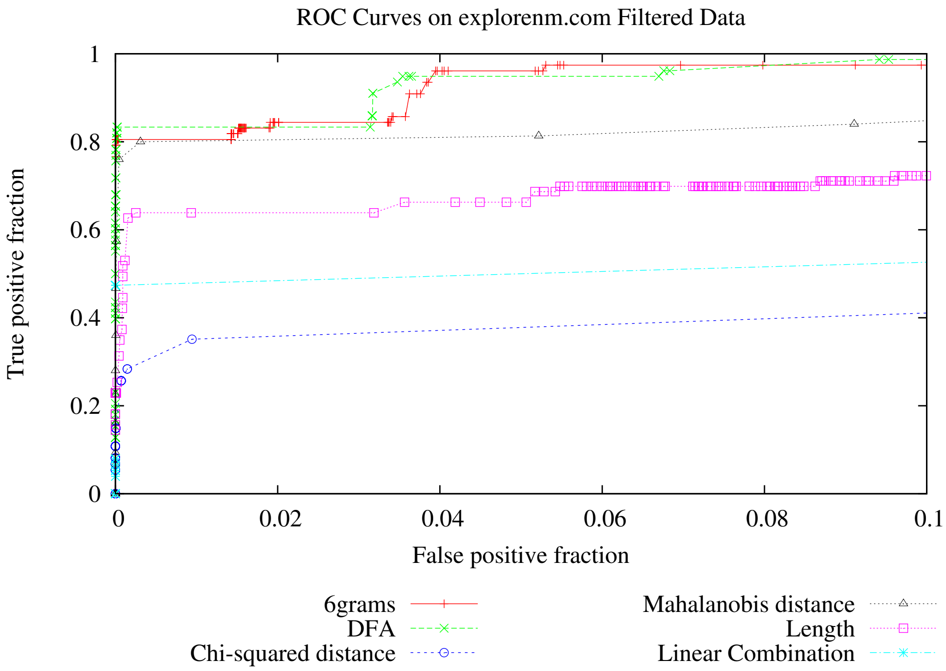


Figure 5.15: Receiver Operating Characteristic curves showing the relative accuracy of the DFA, 6-grams, χ^2 distance between the idealized character distribution and test distribution, Mahalanobis distance between character distributions and the length algorithms when tested on explorenm.com data.

5.4.9 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 5.8 shows the false positive rate per day, assuming a true positive rate of only 80% is required. This table presents data for training on all three cs.unm.edu data sets. This table shows that only the 6-grams and the DFA trained on filtered data has an acceptable false positive rate for a web site like the UNM CS department. These data do not include the grouping potential attacks heuristic; see Section 5.5 to see that heuristic applied.

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 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. This affect appears in Tables 5.1, 5.2, and 5.3. 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 [102].

5.5 Effect of heuristics

The heuristics I developed (Section 3.10) improve accuracy and save model space. Heuristics are required for the DFA and n-grams to be accurate, and in some cases are required to allow the algorithm to run without exhausting the machine's available memory. Section 5.5.1 shows accuracy of these algorithms with and without heuristics, while Section 5.5.2 shows the effect of the heuristics on the size of the model.

Algorithm	**FP/day (filtered)**	**FP/day (unfiltered)**	**FP/day (unfiltered + Code Red)**
Mahalanobis distance character distribution	91,524	56,194	91,631
χ^2 of idealized character distribution	∞	∞	∞
Length	∞	∞	∞
	∞	∞	∞
3-grams	348	4,942	2,170
4-grams	989	3,465	2,580
5-grams	732	6,738	4,958
6-grams	13	6,571	4,693
7-grams	32	15,349	8,806
DFA	37	4,241	1,178
Markov Model	91,675	91,675	91,675
Markov Model (log transform)	39,824	39,807	39,812
Linear combination	∞	∞	∞

Table 5.8: False positive rate per day for the various algorithms, trained on the cs.unm.edu 2004-11-12 and tested on cs.unm.edu 2004-11-17 data, and assuming that a true positive rate of 80% is required. Algorithms marked with ∞ did not achieve this rate. The log transform version of the Markov Model is after the data was transformed as described in Section 5.4.4.

5.5.1 Heuristics and accuracy

Figure 5.16 shows that the combination of heuristics makes the difference between an unusable algorithm and an accurate algorithm. Both the 6-grams and DFA significantly undergeneralize, and hence produce so many false positives that they would be off the scale using the scale on the ROC plots elsewhere in this chapter. With the addition of the generalizations provided by the heuristics, the algorithms are the most accurate of those I tested.

5.5.2 Heuristics and structure size

Some of the heuristics reduce the resulting structure size; this reduction in structure represents generalization. To show the effects of the heuristics (Section 3.10), each algorithm was trained on the cs.unm.edu 2004-11-12 data and tested on the cs.unm.edu 2004-11-17

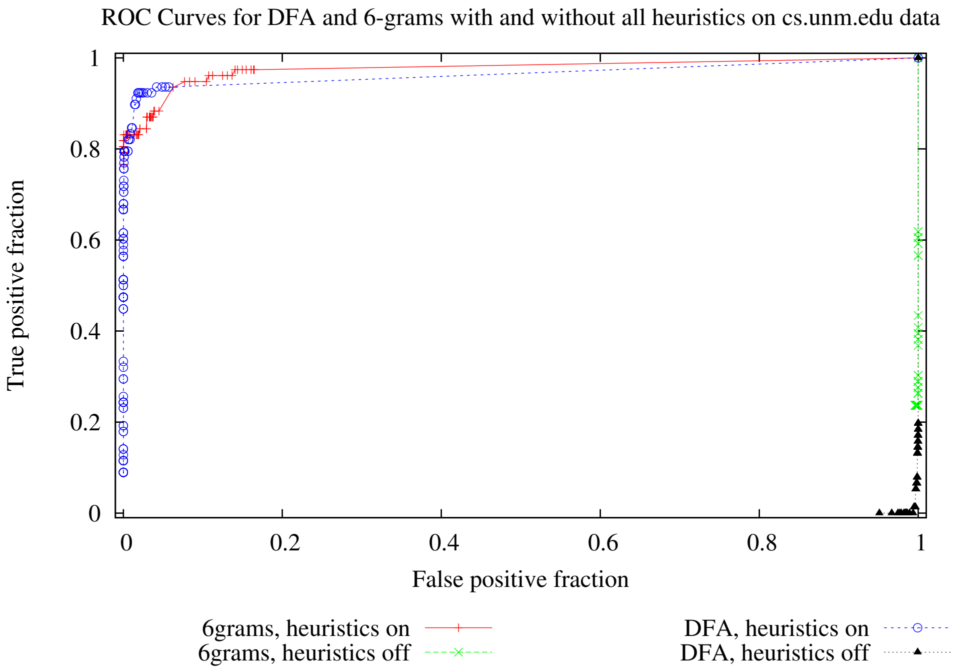


Figure 5.16: Receiver Operating Characteristic curves showing the difference between the default heuristic settings and results with the heuristics turned off.

data with the heuristics all held constant at the default values (Table 3.1 showed the default values) except for the heuristic(s) being tested. These tests were performed only on DFA induction and 6-grams; these algorithms are the only two using tokens.

Table 5.9 shows the number of 6-grams and size of the DFA with the default heuristics on, as well as with individual heuristics turned off; this approach for presenting the data is required because the 6-grams exhausted the system memory on the test machine under some of the tests required to present the data by showing the result of adding individual heuristics. Each heuristic makes at least some difference in the 6-grams and DFA memory requirements, with the file name and type heuristic making nearly a factor of two difference in number of 6-grams and DFA edges. For 6-grams, this heuristic makes the difference between having a model that fits into memory and one that does not.

Heuristic disabled	6-grams	Nodes	Edges
Default	498,211	9,164	67,436
file type and name	845,957	9,427	102,767
IP and host name	542,838	11,095	82,902
dates	665,328	9,208	68,229
hashes	525,609	9,237	70,920
q-values	500,339	9,320	69,452

Table 5.9: Structure sizes for 6-grams and DFA with individual heuristics turned off. The default line shows the state when the heuristics are at the default settings (all of the listed heuristics on).

5.6 New data tests

The results presented earlier in this chapter do not come from my having so finely tuned the heuristics and algorithms to the data. To show this, I took a new data set from pugillis, cleaned it, and ran the DFA and 6-grams on this data. The results in Figure 5.17 for the DFA and in Figure 5.18 for the 6-grams. In both cases, the accuracy for all three sites was similar to the accuracy results presented earlier in this chapter.

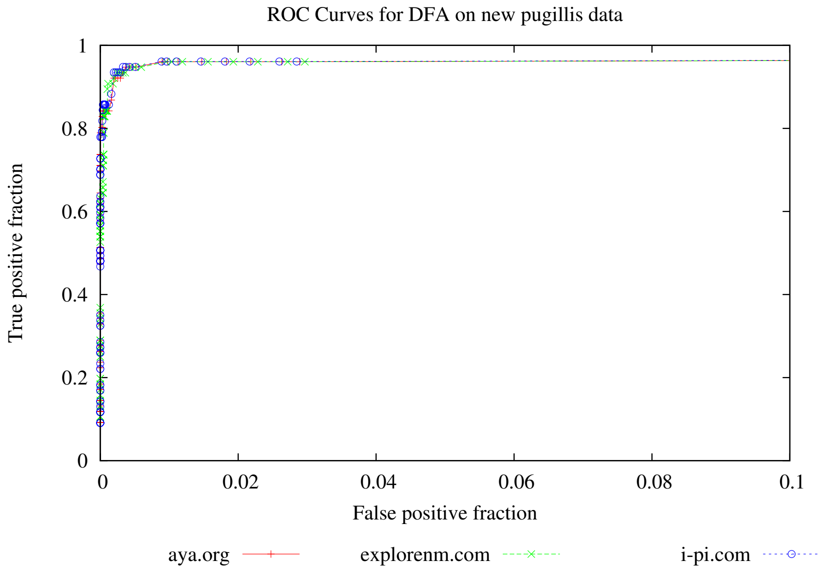


Figure 5.17: ROC curve for DFA on the new pugillis data.

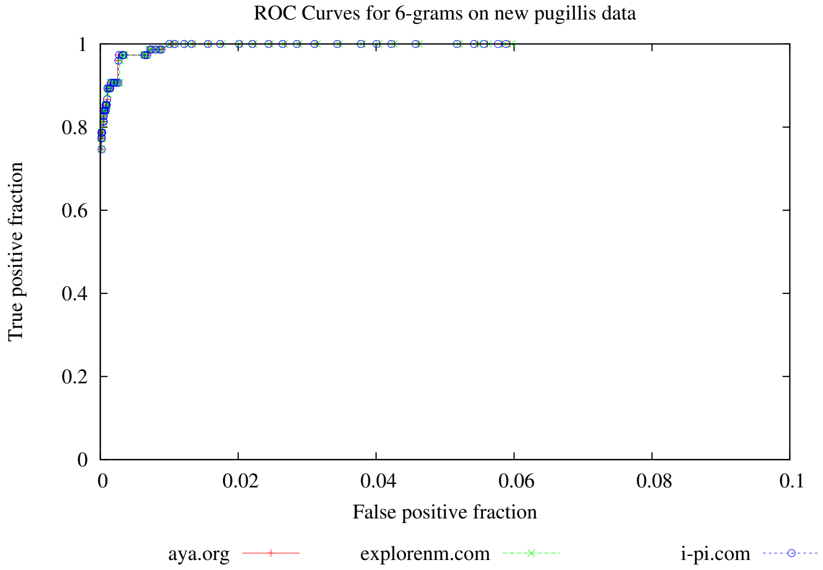


Figure 5.18: ROC curve for 6-grams on the new pugillis data.

Chapter 6

Discussion

Many of the results in Chapter 5 can be explained by the generalization performed by the algorithm: too much, too little, in the wrong place, etc. To delve further into this idea, this chapter looks at the size of the set represented by the model of normal induced by the algorithm, and also at the portion of this set that is legal HTTP. Ideally, we want the actual set size, but the sets discussed in this chapter are all infinite. To deal with this problem, I compare the rate of normal set growth with the request length. These results lead to an explanation for the relative accuracy reported in Section 5.4. Beyond an explanation for the results, algorithm accuracy can be predicted, and this prediction allows comparison of algorithms. Section 6.1 provides details.

Generalization also explains why the heuristics described in Section 3.10 aid accuracy. Beyond explanations, some models allow automatic identification of undergeneralizing regions where additional heuristics are needed. Section 6.2 covers these topics.

Another explanation for part of the results in Chapter 5 can be explained by the differences in the models used by the algorithms—statistics on characters versus learning the lexical structure. Section 6.3 discusses this difference.

Applying the observations about generalization and the data model(s) used by the algorithms, Section 6.4 looks at all of the algorithms and explains where they are accurate, and where the accuracy suffers.

The properties of HTTP described in Section 2.4 affect anomaly detection. Section 6.5

contains this discussion, while Section 6.6 discusses the consequences of allowing attacks in the training data.

6.1 Normal set size, generalization, and accuracy

In the past, when a researcher went to apply anomaly detection to a problem, she would use her knowledge of models and anomaly detection algorithms to hypothesize a combination that would provide acceptable accuracy and performance. If the resulting combination was close enough, she might tweak the model or algorithm using various heuristics, again guided by her experience and intuition. This section describes an approach for comparing relative algorithm generalization.

To choose the anomaly detection algorithms still requires experience with models and algorithms; the model must accurately represent the data. Controlling generalization is necessary, but not sufficient, for accurate anomaly detection—a representation of an HTTP request as the number of 0s in the bitstring encoding of the request is unlikely to provide accurate anomaly detection, regardless of the generalization. For protocols such as HTTP where the fields exhibit different variability, the model must be able to distinguish between the different portions of the data.

The goal of an anomaly detection system is an accurate model of the data instances considered normal—the normal set. If the algorithm over- or undergeneralizes, accuracy problems may result (Section 2.4.6). One measure of the algorithm's generalization is the size of the normal set. If two algorithms (A and B) are trained on the same data, yet A accepts more instances as normal than B, then A generalizes more than B.

When calculating the size of the normal set for each model, if the input is bounded, the set size is bounded. Systems without a bound on the input length (e.g., HTTP), will have an infinite normal set. In this case, the calculation is of the growth of the normal set size with increasing input size. Comparing the set sizes (or growth rates) allows a comparison of generalization; the algorithm with the larger normal set size (faster growth rate) generalizes more. For some of the HTTP anomaly detection algorithms, these results are provided later in this section.

A better guide for the amount of overgeneralization comes from the portion of the normal set that represents legal input for the protected system. For HTTP, this is the portion of the normal set representing legal (or, in actuality, commonly-used) HTTP. The non-legal portion of the set represents overgeneralization—it should never occur in expected instances—a set with many irrelevant entries allows an attacker more latitude for mimicry attacks. As a simple example, an anomaly detector using the length of the request in characters considers all strings of the appropriate length normal, regardless of whether the string represents a legal HTTP request. The information on the portion of normal which is legal provides information about the extent of the algorithm's overgeneralization. For HTTP, this analysis is more involved than the size or growth rate of the normal set.

For the following discussions, I make use of the definitions and notation in Table 6.1. The data set I use for sizes and lengths is the cs.unm.edu 2004-11-12 data. When calculating the fraction of the set representing normal requests, an important fact is number of HTTP methods expected. In this data set, most requests use the `GET` method, with a few using `PROPFIND`.

6.1.1 Length

The generalization done by the length measure is to accept any string having the proper length. For the simple case where only one length is accepted, the generalization set is all strings of length l. Since $\mid \Sigma \mid = 256$ for web servers, there are 256^l different requests of length l. If an exact match is not required (e.g., if one standard deviation is acceptable, the range is $l \pm \sigma$), then additional generalization occurs. In this case, the anomaly detection system accepts requests of lengths l to m, and the resulting set size is:

$$\sum_{i=l}^{m} 256^i = 256^l \sum_{j=0}^{m-l} 256^j = 256^l \frac{256^{m-l+1} - 1}{255} = \mathcal{O}(256^m).$$

Of the set of accepted strings, note that only a small fraction are legal HTTP requests. Of the strings of length l, $\frac{256^{l-4}}{256^l} = \frac{1}{256^4} \approx 2.3 \times 10^{-10}$ is the fraction that begin with `GET` followed by a space, how a request using this method starts. In other words, almost every string in the accepted set is an illegal HTTP request.

A	The adjacency matrix for a directed graph.
a_{ij}	The element (i, j) in A.
${a_{ij}}^{(k)}$	The element (i, j) in A^k.
b_i	When a frequency is binned, b_i is the value in the ith bin.
D	A directed graph.
E	The set of edges from the directed graph.
f_c	The relative frequency of character c.
F	A relative frequency distribution, ordered from largest to smallest.
$F(i)$	The ith element in the frequency distribution F. Note that f_i is not necessarily $F(i)$.
g	A single n-gram—a sequence of n characters or tokens; $g \in N$.
$\mathcal{ICD}$	An Idealized Character Distribution—a list of 256 relative frequencies, $\sum_1^{256} f_i = 1$. See Section 3.2 for details.
i, j, k	Integer indices.
l	The length of an input string. In the context of web servers, this value represents the length of a request. In some of the analyses, l is the maximum length of a request.
m	A length of an input string; $m > l$.
n	The number of sequential characters from the training data in an n-gram.
N	The set of n-grams generated from the training data.
p	The number of vertices in a graph.
S	The subset of n-grams or nodes in the DFA where the (first) token is a valid HTTP method.
v	A vertex in a graph.
V	The set of vertices in a graph.
z	The number of non-zero relative frequencies in an $\mathcal{ICD}$.
Σ	The character set for the algorithm. In web server data, Σ consists of 8-bit ASCII for algorithms working with characters, and the set of possible tokens for the algorithms that use tokens.
σ	A standard deviation.

Table 6.1: Mathematical definitions and notation used in this chapter.

6.1.2 χ^2 distance between character distributions

As one of their measures, Kruegel and Vigna [146] defined an idealized character distribution ($\mathcal{ICD}$), where the relative character frequencies were sorted from most to least frequent. Section 3.2 contains a full description of this measure.

To calculate the number of strings that an $\mathcal{ICD}$ represents, we need to look at the changes that occur to a request before the decision is made concerning whether or not the request is normal:

1. The relative character distribution of the request is calculated.
2. The distribution is placed into six bins
3. The χ^2 test indicates whether the binned distribution matches the $\mathcal{ICD}$.

Note that each step produces an infinite sets if no restrictions are placed on the length of the input. In practice, the input is a finite length, l.

To go from a request to a distribution, note that this model does not make use of which character has a given frequency, only what the frequencies are. The tail of the distribution might (and, in real data, will) consist of several zeros. Let z be the number of non-zero entries in a frequency distribution. Note that $z \leq 256$ due to the size of the 8-bit character set. For the `passwd` example from Section 3.2, $z = 5$.

Because the frequency distribution does not specify the characters associated with each value, one distribution represents all of the permutations of 256 characters that can be used to generate a matching distribution using z different characters. The number of such permutations is:

$$\frac{256!}{(256-z)!}. \tag{6.1}$$

Once we have chosen the characters that give us a distribution, next we consider how these characters might be arranged. Because this model does not consider the order of characters in the request, we need the number of arrangements of z distinct characters in a string of length l:

$$\frac{l!}{\prod_{i=1}^{z}(f_i l)!} \tag{6.2}$$

where $\sum_{i=1}^{z} f_i = 1$ and $f_i l$ is an integer for each i.

In the measure as described in Section 3.2,

bin number	1	2	3	4	5	6	
holds	1	3	3	5	4	240	**values.**

When placing a distribution into bins, note that multiple distributions produce the same binned result. For example, $F_1 = (\frac{3}{4}, \frac{1}{4}, 0, \ldots, 0)$ and $F_2 = (\frac{3}{4}, \frac{1}{8}, \frac{1}{8}, 0, \ldots, 0)$ both produced the binned representation: $(\frac{3}{4}, \frac{1}{4}, 0, 0, 0, 0)$. We want the number of different distributions that produce the same bin values.

Consider if we had 256 bins instead of six. Then, the factors (6.1) and (6.2) from above determine the number of binned representations. Now, suppose that we have 255 bins, where the last two frequencies are summed to provide the value in the last bin, b_{255}. Note that $b_i = \frac{t}{l}$ for t an integer such that $t \in [0, l]$. Hence $t = lb_i$. The number of distributions that result in the same binned representation is the partitions of the integer t into two or fewer terms. If the last three frequencies were summed to produce b_{254}, then we need the partitions of lb_{254} into three or fewer terms.

Define $p(i, j)$ as the number of partitions of i into j parts where order matters. In our case, the order is from largest to smallest matching the frequency distribution ordering. From [41],

$$p(i, j) = \binom{i+j-1}{i} = \frac{(i+j-1)!}{i!(j-1)!} = \frac{\prod_{k=1}^{j-1}(i+k)}{(j-1)!}.$$

Therefore, assuming $z = 256$, the factor of expansion provided by Kruegel and Vigna's binning the frequency distribution is:

$$\begin{aligned}
B &= 1 \cdot p(lb_2, 3)p(lb_3, 3)p(lb_4, 5)p(lb_5, 4)p(lb_6, 240) \\
&= \frac{\prod_{k=1}^{3-1}(lb_2+k)}{(3-1)!} \frac{\prod_{k=1}^{3-1}(lb_3+k)}{(3-1)!} \frac{\prod_{k=1}^{5-1}(lb_4+k)}{(5-1)!} \frac{\prod_{k=1}^{4-1}(lb_5+k)}{(4-1)!} \frac{\prod_{k=1}^{240-1}(lb_6+k)}{(240-1)!} \\
&= \frac{\prod_{k=1}^{2}(lb_2+k) \prod_{k=1}^{2}(lb_3+k) \prod_{k=1}^{4}(lb_4+k) \prod_{k=1}^{3}(lb_5+k) \prod_{k=1}^{239}(lb_6+k)}{2!2!4!3!239!} \\
&= \frac{(lb_2+1)(lb_2+2)(lb_3+1)(lb_3+2)(lb_4+1)(lb_4+2)(lb_4+3)(lb_4+4)(lb_5+1)(lb_5+2)(lb_5+3) \prod_{k=1}^{239}(lb_6+k)}{2 \cdot 2 \cdot 24 \cdot 6 \cdot 239!}
\end{aligned}$$

$B \geq 1$, and B is an upper bound because order matters, i.e., because of the binning, $3+1+1$ and $1+3+1$ are distinct. In most real-world cases $z < 256$.

Next, this measure uses χ^2 to compare the $\mathcal{ICD}$ of the training data with the binned distribution frequencies from the request under consideration, calculating a probability that the two distributions are the same. Since an exact match is not required, additional distributions are accepted as normal. This factor will only make a bad growth rate worse, so let Y be the factor that this generalization introduces, and note that due to the construction of the $\mathcal{ICD}$, there exists at least one request with a distribution matching the $\mathcal{ICD}$. Therefore, $Y \geq 1$.

The size of the set represented by the model of normal is at least:

$$\left(\frac{256!}{(256-z)!}\right)\left(\frac{l!}{\prod_{i=1}^{z}(f_i l)!}\right)BY = \frac{256!l!}{(256-z)!\prod_{i=1}^{z}(f_i l)!}BY$$

As with the length measure, all but a miniscule fraction of this set is composed of strings that are not legal HTTP.

6.1.3 Directed graphs

Both n-grams (Section 3.8) and DFAs (Section 3.9) represent directed graphs, and both will need the following analysis.

A directed graph, D, can be represented as an adjacency matrix A with elements a_{ij}, where $a_{ij} = 1$ when an edge goes from i to j, and 0 otherwise. Let V be the set of vertices for D, and let $p = \mid V \mid$.

The number of walks of length k from v_i to v_j in D is ${a_{ij}}^{(k)}$ ([28], page 13). In the worst case, a complete graph, $a_{ij} = 1$ for $1 \leq i, j \leq p$. Then A consists of all 1s, and $A^2 = pA$. By induction, $A^k = p^{k-1}A$, so ${a_{ij}}^{(k)} = p^{k-1}$.

The total number of walks of length k in D is the sum of all the elements in A^k; in the worst case, this is

$$\sum_{i=1}^{p}\sum_{j=1}^{p} a_{ij}^{(k)} = \sum_{i=1}^{p}\sum_{j=1}^{p} p^{k-1} = p^{k+1}.$$

For the worst case directed graph, the effects of restricting the start and/or end nodes of the walks are:

Number of Walks	**Walk restriction(s)**
p^{k-1}	Specified start and end nodes
p^{k}	Specified start node, unspecified end node
p^{k+1}	Unspecified start node, unspecified end node

n-grams n-grams generalize by not listing the entire sequence of letters in the input string, but instead by requiring only that every sequence of length n be in the set N of n-grams induced from the training data (Figure 3.1 in Section 3.3). In contrast with the

DFA, no distinguished starting or ending point exists, so the size of the accepted set is the total number of walks in the directed graph. From the earlier analysis, for requests of length l, the worst-case value is p^{l+1}.

The portion of this set that is legal HTTP will be approximately the walks of length l from the $g \in S \subset N$ representing the n-grams where the first token is the HTTP method. The fraction of the set that is legal HTTP will be approximately $\sum_{|S|} p^l << p^{l+1}$, as the sum will be over paths with specified start nodes from the set S, but the end nodes are unspecified.

Note that as I used it, this algorithm works with tokens, not characters; most tokens consist of multiple characters. Therefore, this set would be smaller than the sets for the algorithms making use of characters.

DFAs A DFA is a directed graph where a node, v_{START}, is identified as the starting state. Valid strings will result in the DFA ending in an accept state. For the analysis, we need a single accepting state. A DFA with more than one accepting state can be easily transformed into one with exactly one accepting state by the following procedure:

1. Add a new symbol, ω to Σ.
2. At the end of the input string, add ω.
3. Add a new node, v_{ACCEPT}, to the DFA. v_{ACCEPT} will be the only accepting node in the transformed DFA.
4. From all of the accepting nodes in the original DFA, add a transition on ω to v_{ACCEPT}.

This transformation will accept the same strings as the original DFA.

For input of length l, the size of the accepted set is the number of walks of length $l+1$ ($+1$ because of the process to transform a DFA with multiple accept states into one with one accept state) starting at v_{START} and ending at v_{ACCEPT}. This value is $a^{(l+1)}_{\text{START,ACCEPT}}$. The number of vertices in a DFA as I implemented it is $\leq$ the number of distinct tokens seen in training. In a DFA with p nodes, the maximum number of walks for an input of length l from v_{START} to v_{ACCEPT} is $p^{(l+1)-1} = p^l$.

This analysis assumes the worst case of a complete graph. In practice, the DFA has substantially fewer edges than a complete graph. In a complete directed graph, every node has an edge going to v_{START}. In the DFA, no node has an transition to v_{START}. Additionally, transitions from v_{START} exist only for the HTTP methods seen in the training data. For the cs.unm.edu 2004-11-12 data, the DFA has two edges exiting v_{START}. Therefore, analogous with the case for n-grams, a more realistic worst case is $\sum_{|S|} p^{l-1} << p^l$ where the paths start from a small set of nodes S pointed to by v_{START}.

Similarly, only a subset of the nodes in the graph have edges going to v_{ACCEPT} that has no outbound edges. Other restrictions provided by the structure of HTTP also limit the connectivity. Due to the construction of the DFA from requests that are (presumably) legal HTTP, all of the accepted set is legal HTTP.

Note that as I used it, this algorithm works with tokens, not characters; most tokens consist of multiple characters. Therefore, this set would be smaller than the sets for the algorithms making use of characters.

6.1.4 Comparing algorithms via normal set growth rate

In the presence of unbounded input, all of the sets accepted are infinite. However, Table 6.2 shows the growth of the set sizes with the growth of the size of the input requests, l. To make these values more concrete, the order of the resulting set sizes in Table 6.3 are calculated given the values from the cs.unm.edu 2004-11-12 filtered data set. Let T be the number of distinct tokens in the training data; $T = 45,654$ in the example data set. Then for the DFA, $p \leq T$; if no compacting of the DFA was performed, then $p = T$. For n-grams,

$$p = \mathcal{O}(\begin{pmatrix} T \\ n \end{pmatrix}).$$

For 6-grams,

$$\begin{pmatrix} T \\ 6 \end{pmatrix} \approx 10^{26} >> 392,048$$

These results explain the different values of p in Table 6.3.

Of the sets accepted by the algorithms, the DFA consists of only legal HTTP requests,

Algorithm	**Unit**	**Growth rate**
Length	character	$\mathcal{O}(256^l)$
χ^2 distance	character	$\geq BY\frac{256!l!}{(256-z)!\prod_{i=1}^{z}(f_i l)!}$
n-grams	token	$\mathcal{O}(p^{l+1})$
DFA	token	$\mathcal{O}(p^l)$

Table 6.2: Growth of the normal set sizes compared with the input length.

Algorithm	**Parameter values**	**Order of normal set size**
Length	$l = 343$	10^{839}
χ^2 distance	$l = 343$; $z = 94$	$\geq 10^{843}$
6-grams	$l = 41$; $p = 392,048$	10^{239}
DFA	$l = 41$; $p = 7,843$	10^{160}

Table 6.3: Orders of normal set sizes using the results from the cs.unm.edu 2004-11-12 filtered data. Parameters come from Tables 4.2, and 5.7 and Figure 5.4. The length is the mean length, and z and the frequencies for the product for the χ^2 distance comes from the $\mathcal{ICD}$ induced from the training data.

whereas the length and χ^2 distance between character distributions only have a miniscule portion of the accepted set that is legal.

These results predict that order of algorithms from fewest to most false positives is: χ^2 distance, length, n-grams, DFA. The difference in growth rates implies that gap between n-grams and length should be easily distinguished. Figure 6.1 shows the result of testing the algorithms without heuristics on the cs.unm.edu 2004-11-17 filtered data. The false positive rates show that the DFA and n-grams substantially undergeneralize. The low true positive rates for the χ^2 distance and length show they overgeneralize. The large difference in growth rates between the χ^2 distance and the DFA is visible in the difference in generalization as shown by the true and false positive fractions of the algorithms.

Calculating the growth rate of the normal set works to coarsely predict the relative accuracy of an anomaly detection system for HTTP. However, this approach will not always work. Consider a system with generalization as shown in Figure 6.2. This anomaly detection system both under- and overgeneralizes, and its normal set size does not predict this behavior.

When a system over- or undergeneralizes too much, the potential of an algorithm might

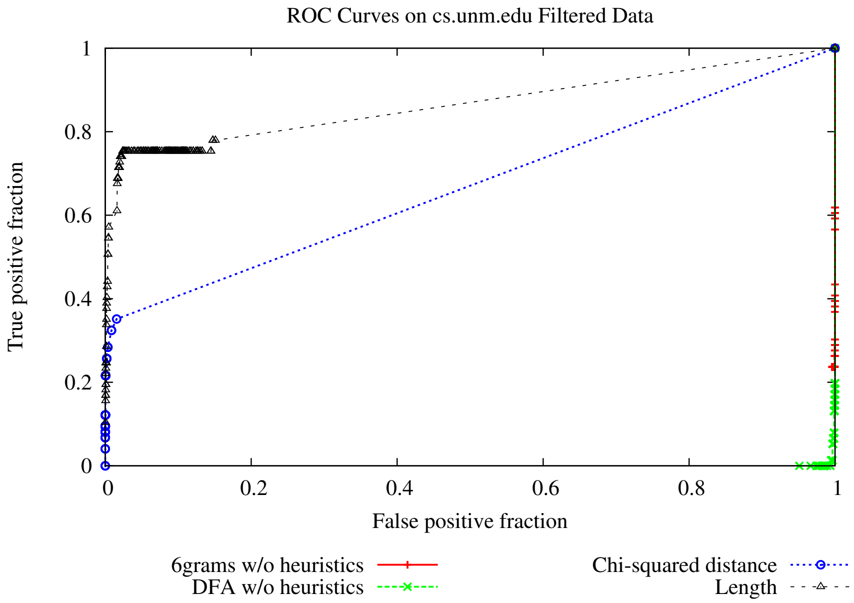


Figure 6.1: ROC curves showing the accuracy of 6-grams, DFA, Chi-squared distance, and Length. Note that these results are for no generalization heuristics enabled.

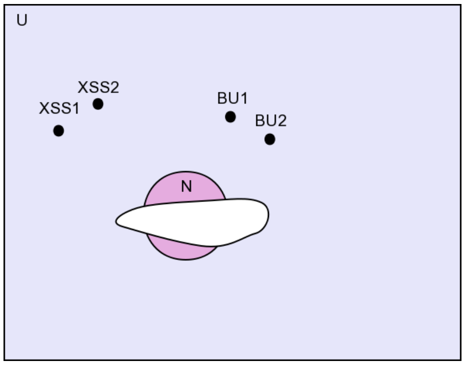


Figure 6.2: A representation of an anomaly detection system that both under- and over-generalizes.

be easy to miss, as is shown by the difference between the DFA or 6-gram algorithms without and with the heuristics applied. Clearly, hitting the "sweet spot" of generalization is critical.

Values from real data show that the actual normal set size can be substantially below that of the upper bound. Calculating the average case growth rate is left as future work. Additional future work includes calculating a more accurate fraction that is legal HTTP and comparing its predictive power with that of the normal set growth rate.

6.2 Heuristics

The difference between an undergeneralizing algorithm and one with controlled generalization added is substantial, and can make the difference between an algorithm with a too-high false positive rate and one that might be usable in a production system. Therefore, controlling generalization is necessary for algorithm accuracy. Heuristics adding targeted generalization are one solution to this problem, and in some cases, they make the difference between whether an algorithm can be used or not. Several of these heuristics can be automatically identified by locating nodes in the directed graph where the out degree is high and the usage count for each link is low—i.e., the algorithm is attempting to memorize highly-variable data values. Section 6.2.1 describes this process.

One effect of the heuristics is to reduce the amount data, n, that must be stored by the DFA or n-grams. The memorization of values can be represented as an expression:

$$n = p_1 p_2 \cdots p_m$$

where each of the p_i refers to a specific portion of the HTTP protocol (e.g., an IP address). Generalization, such as replacing an IP address by a token indicating if it is well-formed changes the corresponding p_i to a small constant; in the IP address example, this value is 2 (well-formed, not well-formed). These effects are visible in Table 5.9.

The heuristic of grouping potential attacks (described in Sections 2.2.2 and 3.10) can be successfully applied to HTTP requests. However, it may not work for a production web server. The administrator may have only two choices: allow possible attacks to proceed (only one request is needed to exploit a vulnerability) or block the unusual requests,

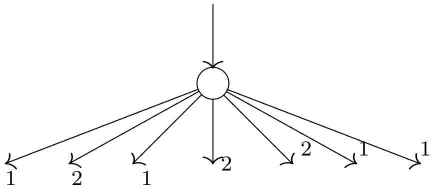


Figure 6.3: An example node in a directed graph where the out degree is high and the usage counts are low.

behavior that will reduce e-commerce site sales. Neither of these options is likely to be acceptable. However, other approaches, such as checkpointing the server (e.g., with a virtualization system such as Xen [20]) may allow a secure method of rolling back to a secure state after delivery of an abnormal HTTP request. Following up on this idea is future work.

6.2.1 Identifying undergeneralization in the model

This section discusses starting with an undergeneralizing algorithm and adding targeted generalizations. This approach stems from experience with improving undergeneralizing algorithms; reducing overgeneralization is a topic for future research. Presumably, you could start with an overgeneralizing algorithm and add restrictions to the generalization. Further work in that direction is left as additional future work.

The model produced by an undergeneralizing algorithm will memorize values when it encounters high variability data. Therefore, to identify these locations, inspect the model, looking for signs of memorization. The exact form these signs take depend on the model. For example, in a directed graph, the pattern indicating simple memorization is a node with a high out degree and low usage counts on all of the outbound edges. For an example of this structure, see Figure 6.3. In particular, a good metric is $\frac{o}{m}$ where o is the out degree of the node and m is the highest usage count of any outbound edge. Sorting the nodes by this value highlights areas of the protocol where additional generalization would provide the biggest benefit.

Once the locations with high-variability are identified, the next step is to develop heuristics. Automatically generating heuristics is left as future work; this dissertation shows applying this process with manually generated heuristics.

Application to HTTP

The goal the automatic identification of undergeneralization locations is to find the locations where the DFA (or other algorithm) is memorizing a set of values. Using the metric described above on the cs.unm.edu 2004-11-12 data with no heuristics, the following were identified as areas where generalization would be useful, in the order of most to least useful:

1. IP address
2. The Referer header field; this is a URL that led the user to the web site, and frequently is the result of a search. The search URLs are complex and highly variable.
3. Integer ranges (used for saving bandwidth by only sending the portions of the resource changed from a cached version).
4. User agent additional information (e.g. user agent version, capabilities, etc).
5. hashes (Entity tags, session IDs, etc)
6. dates
7. host names

Note that this approach identified several locations where generalization was needed that were the same as those I found while working with the data (Section 3.10). It also found the Referer header field, which, in hindsight, is an obvious candidate for generalization.

The file name length heuristic was not discovered by the automatic process because a given directory has a small, finite number of files. Therefore, the out degree was not high enough in any of the directories on the cs.unm.edu web site to show up in the automatic identification process.

The floating point numbers in q-values were not identified by the automatic process simply because in the training data, only a small number of distinct floating point values appeared, and memorization of these values was not causing accuracy problems.

One effect of the generalization heuristics is to reduce the problems caused by the nonstationary nature of the HTTP requests. Since the current date and time are constantly changing, replacing a portion of the request that changes with one that does not lowers (but does not eliminate) the nonstationarity of the data. Assuming a valid date cannot be used as an attack, this heuristic helps move the novelty of an attack to a more noticeable value.

This idea could be taken too far; overgeneralization due to applying too many generalization heuristics can cause the system to miss attacks. As it is, modifications by the heuristics explains why two attacks are considered identical by the DFA (Table 5.5).

This approach is not specific to the DFA. The n-grams also result in a directed graph, and the same structure appears in the resulting graph when the n-grams are trying to memorize a high-variability region.

6.2.2 Heuristics and the normal set

The generalization heuristics described in Section 3.10 increase the size of the normal set for the DFA and n-grams. Table 6.4 contains the details of the size increase. The heuristics **file types only**, PHP Session IDs, and **recognize host names** enlarge the set a great deal. The values in Table 6.4 are normal set size growth; because of loops in the DFA, it accepts an infinite number of HTTP requests. Therefore, these values are not directly comparable to those in Table 6.3, which are based on the growth rates and not absolute set size.

The effect on normal set size of these generalizations show that for an undergeneralizing anomaly detection system, which additional generalizations are applied is critically important. For example, the PHP session ID generalization makes the normal set substantially larger. Assuming the properties of a cryptographic hash are met, these session IDs represent a 128 or 160 bit random string. This portion of the HTTP request is better matched by a length and character distribution.

Generalization	Replacement set size
recognize dates	all dates with a two- or four-digit year: $366 * 10,000 + 366 * 100 = 3,696,600$; all possible times: $24 * 60 * 60 = 86,400$; seven days; 54 time zones (non-standard, but common). The resulting set size is $120,727,998,720,000 \approx 10^{15}$
file types only	all legal file names; up to 251 characters (at least four are specified in the type) with 254 possible values $251^{254} \approx 10^{610}$
PHP Session IDs	2^{128} if MD5, 2^{160} if SHA1
recognize IP addresses	2^{32}; a smarter check would only allow class A, B, and C addresses which is $2^{31} + 2^{30} + 2^{29} = 3,758,096,384$
recognize hostnames	Any legal hostname. Host names are not length limited, therefore this generalization allows an infinite number of strings.
recognize q-values	A floating point value in $[0,1)$ with zero to three digits to the right of the decimal place or 1 with zero to three zeros after the decimal place; $1 + 10 + 100 + 1000 + 4 = 1,115$

Table 6.4: The effect of generalization heuristics on the normal set size.

6.3 Characters versus lexical structure

The results in Section 5.4 show that character-based algorithms are notably less accurate than the token-based algorithms. When I began this research, I tried using n-grams on the requests treated as character streams but discarded this approach because it was not accurate.

Tokens represent a higher lexical unit, and are used by the system to represent meaning. Attacks often represent nonsense. With the need to "ship the product yesterday" and other deadlines, programmers often focus on making the system work under normal circumstances and spend fewer resources on nonsense cases. Additionally, to consider all of the ways in which nonsense may be represented requires thinking in ways many programmers were not trained. In my attack database, most, if not all, of the attacks represent nonsense.

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. Obviously, this is affected when the training data contains harmless attacks; Section 6.6 discusses this topic further. Applying these concepts to the algorithms I 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.

The idea of representing the meaning of the request allows me 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.

6.4 Algorithm accuracy

The exact ordering of algorithms by accuracy depends on the acceptable false positive fraction, but the ROC curves of the algorithms are often well-separated throughout the graph. Note that the token algorithms are always better than the character algorithms. The 6-grams and DFA are normally close, and sometimes the Mahalanobis distance is nearby, other times it is substantially worse. The gap between these three and the length and χ^2 distance is usually notable. The rest of this section looks at the algorithms in more detail.

6.4.1 Length

Trained on filtered data, this measure can detect some buffer overflows and cross-site scripting attacks. However, when harmless variants of these attacks are included in the training data, the measure becomes nearly useless. Additionally, some attacks such as the Apache chunked transfer error (Figure 2.9) and some variants of Nimda (Figure 2.5) are short enough that they could easily 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.

6.4.2 Character distributions

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 (Figure 2.9) and some variants of Nimda (Figure 2.5) 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.

Wang and Stolfo [263] tested the Mahalanobis distance using the MIT Lincoln Labs data (Section 2.3.2). This data set contains only four HTTP attacks. In the years since the MIT data were collected, attack characteristics have changed; my more comprehensive attack data set illustrates the effect of this difference on this algorithm (Figure 5.6). Kruegel and Vigna [146] were not as 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. An interesting topic for future work is to investigate why the Mahalanobis distance sometimes does nearly as well as the DFA and 6-grams, and other times is worse than the length algorithm.

6.4.3 Markov Model

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. My 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 [146]. 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 more tokens to work with mean more places where normal diversity results in lower probabilities for any given HTTP request.

Kruegel et al. [146, 147] tried a Markov model for representing the characters in CGI parameters. However, they found that most transitions in their Markov model were rare (i.e., their data had high variability); thus, they instead used their Markov model as a zero/non-zero test to see if the structure of the CGI parameters had been seen in training. By ignoring the magnitude of transition probabilities, they in effect used their Markov model as a DFA. Generalization occurred when the NFA they originally built was compressed through state merging.

6.4.4 CGI parameter algorithms

The CGI parameter algorithms (order described in Section 3.3, presence or absence of parameters from Section 3.4, and enumerated or random parameter values from Section 3.5) are not well-suited for application to the complete request. The order of the HTTP header fields is likely to be fixed for a given browser version, but might change with versions, browsers, and robots. A different order than that seen in the past might indicate an abnormal request, or it could simply mean, for example, that Microsoft has released a new version of Internet Explorer. It would be trivial for an attacker to mimic the order of header lines of a popular browser, so this algorithm is unlikely to provide useful information for identifying attacks. The one attack that might be detected is Apache Sioux (Figure 2.8 and [54]). This attack repeated a header line many times causing the vulnerable version of Apache to run out of memory. In general, by ignoring the values of the headers, this approach will miss almost all attacks when applied to the complete HTTP request. A similar argument can be applied to the presence or absence of header lines.

Applying the enumerated or random parameters test to the header lines of a complete request would be a coarse measure. If a value is determined to be random, then the value provides no information. The nonstationarity of web sites means that new values are common. The difference between a new value and an attack cannot be distinguished by this algorithm alone. Kruegel and Vigna [146] applied this measure where the parameters are more likely to be from a fixed set. This algorithm is more likely to be useful in these cases.

6.4.5 Linear combination

As Kruegel and Vigna [146] implemented the linear combination, most of the algorithms had to agree that the request was normal in order for the combination to consider the request normal. When algorithms such as the character distribution and length overgeneralize as they do, this effect filters out to the combination as well. When Kruegel and Vigna developed this test, they limited its use to CGI program parameters. Used in this manner, The CGI parameter tests (order, presence or absence, and enumerated or random), with their limited to no generalization probably explains much of the accuracy they achieved with their system.

When combining measures such as the character distribution, length, CGI parameter tests, and the Markov model, the resulting generalization depends on how the individual tests are combined. If all models must agree that the request is normal, then the least general will usually be the model indicating an abnormal request, and hence control the result. 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.

6.4.6 Directed graphs

n-grams using tokens model the structure of the request by encoding sequences of tokens, in effect, representing the input as a directed graph. Generalization occurs in two ways. First, for all but the shortest requests, the number of tokens exceeds n, so paths not corresponding to any request in the training data might exist in the graph. Second, by allowing a small number of complete mismatches, additional requests can be accepted. This generalization is limited compared to that performed by the length and character distribution algorithms. The improved true positive rate shows that this algorithm is closer to optimal generalization, and the model can better tell the difference between normal and nonsense.

The DFA induced using tokens is also a directed graph representing the structure of the HTTP request. Generalization occurs in the DFA compaction (see Section 3.9). Additional generalization occurs because paths not corresponding to any request in the training data might exist. Instead, these paths represent mergers of two or more requests. Finally, generalization also occurs when one or more "missed tokens" (Section 3.9.1) are allowed. The generalization is 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.

Comparing the DFA to n-grams, the sequence nature of the n-grams means it does better at constraining resource paths; these appear as sequences of directories separated by the `/` token. Contrast this representation with the DFA, that uses a loop structure to represent the path, and hence will accept permutations and substitutions in the path, whether or not they are valid. This effect shows up in Tables 5.1 and 5.2. For example, if `/a/b/c` and `/d/e/f` are the only valid paths, then with $n \geq 3$, the n-gram algorithm will accept no other paths. However, because the DFA represents paths as a loop of choices (see Figure 6.4 for an example of this loop), the following paths are some of the paths acceptable to it: `/a/b/c`, `/a/a/a`, `/a/e/c`, and `/f/e/d`.

A weakness in both the DFA and n-grams algorithm is that if the attack can be presented in a small number of tokens, the algorithm cannot tell the difference between a

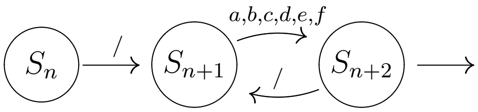


Figure 6.4: Example portion of the DFA encoding paths. Note the loop between nodes S_{n+1} and S_{n+2}.

novel request with a few new tokens and an attack. This issue 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 (Figure 2.6) used a multitude of /s to cause an out-of-memory condition in an older version of Apache.

6.5 HTTP

6.5.1 Diversity and variability allowed by the protocol

Section 5.1.1 showed that some algorithms can distinguish between web sites. The DFA and 6-grams algorithms never perform worse on the site on which they trained. A contrast is the Mahalanobis distance algorithm that for the cs.unm.edu data is more accurate on a different web site. Had the length and χ^2 distance algorithms been sufficiently accurate to have been included in these results, they also likely would have experienced difficulties. Considering where the differences exist in the requests, this result is unsurprising—most people choose their path names from a set of characters consisting primarily of alphanumerics; the language and file types all are specified in the standard and consist of lower case letters and the −; the browser identity (in all existing browsers and robots in my data set) is composed from only a slightly larger set of characters. The average request length varies only slightly between web sites in my data. An IDS using Mahalanobis distance, length or χ^2 distance might be vulnerable to a "write once, run everywhere" attack.

Section 5.1.3 shows attacks are more diverse than normal requests. Knowledge of one attack does not necessarily convey knowledge of others. This result suggests that the approaches where researchers have generalized attacks (e.g., [3, 218]) might still miss novel attacks. The generalization of a buffer overflow attack might allow detection of other buffer overflow attacks, but it is unlikely to aid in detecting cross-site scripting attacks.

In general, different resource paths are sufficient for the DFA and n-grams to distinguish between requests. This explains why the non-attack data in Table 5.5 exhibit similarity ≈ 0.3 and not higher.

Section 2.2.1 noted that only one rule-based system had been proposed for HTTP. These systems often attempt to generate a complete set of rules describing the expected behavior—i.e., they do little generalization. Because HTTP is a highly-variable protocol; this might hinder good rule generation by either humans (expert systems) or automatic systems.

6.5.2 HTTP is a nonstationary data source

The results in Section 5.1.2 show that HTTP is a nonstationary data source. Whether or not nonstationarity is a problem depends on the amount of generalization the algorithm performs. An algorithm with minimal generalization or one that is prone to undergeneralizing (e.g., DFA) will be more likely to detect these changes and respond with an increased rate of false positives. Such an algorithm requires a mechanism for adapting as the protected web site changes. Additionally, this adaptation might allow the algorithm to show an improvement over time as it continues to learn from the test data.

The DFA undergeneralizes without heuristics, and as such it is sensitive to even small changes in the web site. However, as Table 5.4 shows, the adaptive techniques described in Section 3.9.1 solve this problem.

The slow degradation of accuracy the 6-grams exhibit implies simply adding new n-grams to the database is insufficient to track the web site. This result is puzzling; adding new n-grams adds new nodes and/or edges to the directed graph, an operation equivalent to adding nodes and/or edges to a DFA. Following up on this result is future work.

The Mahalanobis distance measure generalizes more than the DFA and 6-grams. This extra generalization allows it to accept more diversity in input, regardless of the diversity's origin.

The linear combination needs a second pass across the training data to determine the individual algorithm values for normal. Re-calculating thresholds every time normal changes implies that the linear combination cannot handle nonstationary data, and would therefore need to be changed to remove this limitation in order to achieve good accuracy over the long-term use expected in a production environment.

Beyond the implications for the algorithms, these data also show that the cs.unm.edu web site changed over the short time the data was collected. Any production anomaly detection system will need to track the changing web site or it will need regular retraining. Since no exploits can occur during training (else the system learns harmful attacks as normal), regular retraining reduces the usefulness of the anomaly detection system by providing an attacker with regular intervals when an attack would potentially go unnoticed.

6.6 Attacks in the training data

If the anomaly detection system is protecting an Internet-connected server, then the training data should include attacks to which the server is not vulnerable (Section 2.3.2). Attackers test for old vulnerabilities for years after the exploit was discovered, hoping to find unpatched systems. Worms continue to attack from poorly managed systems[1] long after they were released. An anomaly detection system that reports these harmless attacks will likely be considered a nuisance and disabled by the system administrator.

Two solutions to this problem exist. The first is to use a signature-based detection system to remove harmless attacks from both the training and real data. This approach might require customization of the signature set for the protected machine, because some network traffic that appears to be an attack at one site might be normal at another. For example, a web site not running WEBDAV [97] would expect all requests using the `PROPFIND` method to be (harmless) attacks against vulnerabilities in Microsoft's WEBDAV implementation or applications utilizing it. However, sites running WEBDAV would need more careful tuning of the signatures. As an additional example, when filtering attacks from my test data (Section 4.3), the *snort* signatures required customization due to differences between the sites and the web applications they used.

[1] Nimda and Code Red were both released in 2001. In late 2004, the cs.unm.edu and pugillis web servers were still receiving these attacks. Neither server was ever vulnerable to these attacks.

The second approach is to include harmless attacks in the training data, allowing the IDS to learn them as part of normal behavior. This approach can be difficult for anomaly detection systems, if the harmless attacks resemble new attacks. Training on data containing attacks is effectively training on data that makes sense as well as nonsense, and trying to build a model that represents both.

To understand why attacks in training data are problematic, consider the example of a HTTP cross-site scripting (XSS) attack. One version of this attack targets a web site that allows comments to be added to a page—for example, most blogs have a way for readers to comment on the blog. The attacker injects code as part of the comment; this code is executed on the browser of the machine that displays the comments. In some circumstances, scripts gain access to cookies, form data, and other browser information that might be sensitive. The scripts can also initiate communication to web servers with additional hostile content.

As an example of this type of attack, suppose the protected system accepts requests of the form:

```
GET /scripts/access.pl?user=johndoe&cred=admin
```

If the `cred=` portion had been vulnerable to a cross-site scripting (XSS) attack in the past, then the training data might include instances such as (the two examples are equivalent, but encoded differently)[2]:

```
GET /scripts/access.pl?user=johndoe&cred=<script>doc
ument.location='http://www.cgisecurity.com/cgi-bin/c
ookie.cgi?' +document.cookie</script>

GET /scripts/access.pl?user=johndoe&cred=%22%3e%3c%73
%63%72%69%70%74%3e%64%6f%63%75%6d%65%6e%74%2e%6c%6f
%63%61%74%69%6f%6e%3d%27%68%74%74%70%3a%2f%2f%77%77
%77%2e%63%67%69%73%65%63%75%72%69%74%79%2e%63%6f%6d
%2f%63%67%69%2d%62%69%6e%2f%63%6f%6f%6b%69%65%2e%63
```

[2] These attack strings make use of examples from `http://www.cgisecurity.com/articles/xss-faq.shtml`.

```
%67%69%3f%27%20%2b%64%6f%63%75%6d%65%6e%74%2e%63%6f
%6f%6b%69%65%3c%2f%73%63%72%69%70%74%3e
```

If attacks such as these occur in the training data, the possible effects for an IDS using statistics describing behavior include:

- The attribute and overall request length would be shifted toward larger values (XSS attacks are longer by the length of the attacking script plus required HTML or related code).
- The attribute character distribution would be biased toward cross-site scripting-type values.
- The requested program (`/scripts/access.pl`) would be unchanged.
- The structure of attributes would be unchanged (`user=` and `cred=` would still both appear together and in the same order).

If a new XSS vulnerability were discovered, this time associated with the `user=` portion, it would likely be accepted as normal by an IDS using character distribution or length for its measure of normal. Similarly, if a new XSS vulnerability were discovered, this time associated with a different program, exploits for it would likely have similar character distributions, attribute lengths, and request lengths to those of the earlier attacks. If these measures were part of a model of normal web requests, the system would be trained to tolerate not only the original attacks but the newer ones as well.

A similar argument applies to other attacks, for example, buffer-overflow attacks. If buffer overflows occur in the training data, the generalization should not accept other buffer overflows as normal (e.g., due to long character sequences, possibly containing machine code, possibly encoded as printable ASCII). If both types of harmless attacks are in the training data, then the IDS should not learn variants and combinations of both attacks as normal.

Chapter 5 shows a single statistical measure of character frequency is insufficient; realizing this, a few researchers have combined multiple statistical measures (e.g., [146]). Combining multiple statistical detectors does not necessarily solve this problem, because

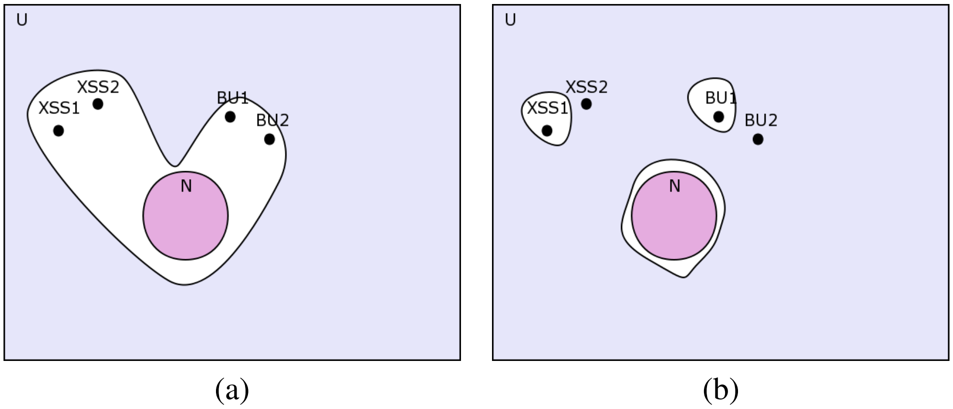


Figure 6.5: Generalization strategies: U is the universe of all possible HTTP requests; N illustrates a hypothetical set of normal requests; XSS1 and XSS2 indicate cross-site scripting attacks; and BU1 and BU2 indicate buffer overflow attacks. If XSS1 and BU1 were in the training data, the lines surround the set of HTTP requests that an anomaly detection system might accept. (a) An overgeneralizing system, for example, a simple statistical system such as character distribution. (b) A more desirable generalization.

each detector must generalize enough to accept the harmless attacks. Only if the combination has the effect of controlling the overgeneralization will it produce acceptable accuracy.

To summarize, if harmless attacks are in the training data, then variants and combinations of these classes of attacks are likely to be accepted as normal by anomaly detection systems that overgeneralize. This situation is illustrated in Figure 6.5(a). Properly controlled generalization is shown in Figure 6.5(b).

6.6.1 The algorithms and harmless attacks in the training data

Length

The length measure overgeneralizes, and when attacks are included in the training data, the generalization is such that the model is unable to distinguish normal requests from attacks. None of the anomaly detections systems described in the literature used this measure in isolation.

Character distributions

If harmless attacks are included in the training data, the model of normal becomes so general that this measure is nearly useless in distinguishing attacks from normal traffic.

The two character distribution algorithms show better accuracy for unfiltered data. One explanation for this unexpected result is that the improvement comes because the attacks in the training data are not in the set of attacks used for testing. If an algorithm has a particular problem with a class of attack, then removing some or all of that class from the test improves the apparent accuracy of the algorithm.

Directed graphs

When trained on data containing harmless attacks, the n-gram learns the position of the attack as well as the attack, reducing the chance that a similar attack in a different location will be accepted as normal. Unfortunately, this is not sufficient, since the 6-grams missed some attacks when trained on unfiltered data. A weakness in this algorithm is that if the attack can be presented in a small number of tokens, the algorithm cannot tell the difference between a novel request with a few new tokens and an attack. This issue was responsible for some of the missed attacks.

When trained on data containing harmless attacks, the DFA also learns the location of the attack relative to nearby tokens, reducing the chance that a similar attack in a different location will be accepted as normal. This effect can be seen by comparing the distance between the curves in Figure 5.11 or 5.10 versus the distance between the curves in Figures 5.5, 5.7, and 5.9.

Chapter 7

Conclusion

Experience with current network servers shows that they are not secure, and therefore additional security measures are necessary for secure operation and, in many cases, compliance with laws relating to privacy and security. One useful security measure is intrusion detection. Three intrusion detection systems architectures exist, each with strengths and weaknesses. Signature systems work well at detecting previously-seen attacks, but they cannot detect novel attacks and they might be subject to crafted false positives. Specification systems can detect novel attacks, but generating a correct specification requires higher skills than writing the program in the first place, and every time the program changes, the specification must be checked for possible updates as well. Anomaly detection systems learn a model from a set of training data; if this model matches the data well, they can detect novel attacks and not require higher skills on the part of the system developer or administrator.

HTTP is an excellent protocol for anomaly detection research. Custom web applications are growing rapidly, but the security knowledge of the programmers of these applications is often weak or missing altogether. An IDS for protecting web servers must deal with the problems associated this area: HTTP is a stateless protocol, web sites change rapidly with time, the HTTP request length is variable length, and the nature of the data means that a one-class learning algorithm is required. Network protocols such as HTTP are especially difficult for anomaly detection, because different portions of the request have different variability.

Harmless attacks are normal for an Internet-connected web server, and an anomaly detection system should not produce alarms on these attacks. If they are contained in the training data, the anomaly detection system has additional generalization issues; it must accept the harmless attacks, while still detecting novel attacks that might be of the same class as other, harmless ones.

We have no good theory of intrusion detection; therefore rigorous testing is essential for knowing which IDS algorithms perform best at HTTP intrusion detection. I tested several previously-proposed algorithms as well as two new to HTTP: n-grams and DFA induction. My test results show that some algorithms are notably more accurate on some web sites. More importantly, my results show that every algorithm missed attacks. This result means additional work is needed to improve the anomaly detection systems.

My test results can be explained by two factors. First is the generalization the algorithms perform. Correct generalization is necessary for accurate anomaly detection. Undergeneralizing systems have problems with false positives, while overgeneralizing systems suffer from false negatives. Length and character distribution are examples of overgeneralizing algorithms, while the n-gram and DFA algorithms undergeneralize if the heuristics adding generalization are not applied. The second factor is the observable chosen for the input data. Building a representation that encodes more of the meaning of an instance allows the algorithm to better distinguish between sense and nonsense. The DFA and n-grams, by using tokens, are examples of this latter approach.

In the past, when a researcher went to apply anomaly detection to a problem, she would use her knowledge of models and anomaly detection algorithms to hypothesize a combination that would provide acceptable accuracy and performance. If the resulting combination was close enough, she might tweak the model or algorithm using various heuristics, again guided by her experience and intuition. My dissertation results improve this situation. Researchers can compare the sizes of the normal set described by the model, and look at the portion of the normal set representing valid data instances. The results of this analysis provides guidance about the amount of generalization and a measure to compare algorithms. This information is an aid in choosing one to fit the problem, and it might also assist in explaining the resulting system accuracy. For an undergeneralizing system, once the algorithm has constructed the model, it can be inspected for locations where additional generalization is required. These locations are where it is memorizing highly-variable results, and targeted additional generalization is needed. The result is that we can now build more accurate anomaly detection systems. Also, the results of my research remove some of the guesswork or intuition previously applied to anomaly detection.

7.1 Contributions

This dissertation advances the state of the art for protecting web servers in several areas. First, it investigates the relationship between generalization and accuracy in anomaly detection, with a focus on identifying techniques for comparing models and predicting relative accuracy. Second, it provides guidance for constructing models where the generalization is appropriate for the data, and automatically identifying locations where additional generalization is needed. Once the undergeneralizing has been localized, heuristics that perform the targeted generalizations based on the structure of the data stream can be applied. With this targeted, controlled generalization, an anomaly IDS can achieve both smaller models (saving memory) and a reduced false positive rate.

These results about generalization grew from manually developing parser and algorithm heuristics to improve anomaly detector accuracy on HTTP requests. The DFA and n-grams with the heuristics applied are more accurate than any previously proposed algorithms. The automatic process identified several of the heuristics I developed, as well as one I missed, the values for the `Referer:` header.

I implemented an open-source framework to test IDS algorithms, and then I used this framework to test nine algorithms to show their performance under realistic conditions. This testing is more rigorous than any HTTP IDS testing reported to date. I am one of the first researchers to test on data containing attacks (unfiltered data). This data is more difficult for an anomaly detection system to use, and my results show the accuracy of the various algorithms under these conditions.

In order to perform the testing of the algorithms, a database of attacks was needed. A good database of HTTP attacks did not exist; therefore I developed the largest open database of HTTP attacks designed for IDS testing. My attack database is the most comprehensive open database of HTTP attacks used for testing HTTP IDSs.

Most previous IDS approaches for HTTP have represented the request as a character string. My work 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 detection detect more attacks than earlier approaches. One reason for this improved performance is that I use 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.

7.2 Moving beyond HTTP

Given the results from working with HTTP, an obvious question is, “where else can this analysis be applied?” My results about generalization and the size of the normal set might apply whenever the anomaly detection system generalizes from training data. Examples of other network protocols that might be amenable to similar analysis include:

SMTP The negotiations between mail servers and the email headers themselves might lend themselves to anomaly detection. The message body itself might have too much variability. Or, it could be that enough structure exists (e.g., language sentence structure, word choices, attached file structure) and anomaly detection can be applied even to the message body.

With an accurate enough model of the email body (or sequences of email messages and their bodies), anomaly detection might prove itself useful in anti-SPAM systems.

SSH, SSL, TLS, IPsec perform complicated negotiations relating to cryptography algorithms. Exploitable bugs in code implementing this negotiation have been found, so anomaly detection might be applied successfully here.

DNS Without the domain name system, the Internet would stop. Several attacks against DNS servers exist. The DNS protocol contains a more structure than HTTP and different variability in different portions of the protocol. The additional structure make make DNS an easier protocol to which anomaly detection might be applied, and the targeted generalizations, similar to those I developed for HTTP, are likely to be useful for this protocol as well.

Beyond computing and networks, the concept of controlling generalization can be applied to other anomaly detection systems. For example, in financial data, "Which transactions are abnormal and possibly represent fraud?"

7.3 Future work

The work presented in this dissertation asks more questions than it answers. An abundance of future research opportunities exist that follow up on my work.

My work showed how to identify locations where additional generalization is needed. It identified several important locations where the DFA was undergeneralizing. It did not identify the **file types only** heuristic, the heuristic with the largest single effect. Additionally, identifying the required generalization is still up to a human expert. Automating this process would be immensely useful.

Many of the results in this dissertation are for adding generalization to an undergeneralizing algorithm. The ability to move the other direction, restricting generalization in an overgeneralizing algorithm, would be useful as well. This problem showed up in my work, since the DFA, an undergeneralizing algorithm overall, overgeneralizes on the resource path. Identifying and automatically finding methods to reduce the generalization would greatly aid system developers.

Using the idea that controlling generalization is necessary for accuracy, interesting future work would include investigating various methods of generalization and ways of combining measures to better model the set of normal requests. The result would be more accurate anomaly detection systems.

Because no algorithm yet provides a good enough attack detection rate for HTTP without an unacceptable false positive rate, we need additional heuristics to improve the algorithm performance. My work can provide guidance for this followup work. Or, it might be that other algorithms would perform better in this environment. With my test framework, testing them requires only an algorithm implementation; the rest of the supporting structure now exists.

My attack database, while the largest open one, does not have representatives of all attack classes. Researchers in intrusion detection need good, standard test data sets (both attack and non-attack) that are current and representative of today's web servers (e.g., e-commerce, simple files, blog sites, etc). For attacks, my attack database is a start, but such a database needs hundreds, if not thousands, of entries. The attacks should be against a server normally running the attacked software to better represent the situation an IDS would face. A larger database would also aid further research into identifying attacks difficult for existing algorithms to detect, and then lead to better heuristics and/or better algorithms.

References

[1] ALESSANDRI, D. Towards a taxonomy of intrusion detection systems and attacks. Tech. Rep. RZ 3366, IBM Research, Zurich Research Laboratory, Säumerstrasse 4, CH-8803 Rüschlikon, Switzerland, Sept. 2001.

[2] ALLEN, J., CHRISTIE, A., FITHEN, W., MCHUGH, J., PICKEL, J., AND STONER, E. State of the practice of intrusion detection technologies. Tech. Rep. CMU/SEI-99TR-028, Carnegie Mellon University, Software Engineering Institute, 2000.

[3] ANCHOR, K. P., ZYDALLIS, J. B., GUNSCH, G. H., AND LAMONT, G. B. Extending the computer defense immune system: Network intrusion detection with a multiobjective evolutionary programming approach. In *Proceedings of ICARIS 2002: 1st International Conference on Artificial Immune Systems Conference* (2002).

[4] ANDERSON, D., FRIVOLD, T., TAMARU, A., AND VALDES, A. Next-generation intrusion detection expert system (NIDES), software users manual, beta-update release. Tech. Rep. SRI–CSL–95–07, Computer Science Laboratory, SRI International, 333 Ravenswood Avenue, Menlo Park, CA 94025-3493, May 1994.

[5] ANDERSON, D., FRIVOLD, T., AND VALDES, A. Next-generation intrusion detection expert system (NIDES): A summary. Tech. Rep. SRI-CSL-95-07, SRI International, Computer Science Laboratory, May 1995.

[6] ANDERSON, D., LUNT, T. F., JAVITZ, H., TAMARU, A., AND VALDES, A. Detecting unusual program behavior using the statistical components of NIDES. Tech. Rep. SRI-CSL-95-06, SRI Computer Science Laboratory, May 1995. `http://www.sdl.sri.com/papers/5sri/`. Accessed 22 August 2002.

[7] ANDERSON, J. P. Computer security technology planning study. Tech. Rep. ESD-TR-73-51, United States Air Force, Electronic Systems Division, Oct. 1972.

[8] ANDERSON, R., AND KHATTAK, A. The use of information retrieval techniques for intrusion detection. In *First International Workshop on Recent Advances in Intrusion Detection (RAID'98)* (1998).

[9] ANON. List of web browsers, 2006. http://en.wikipedia.org/w/index.php?title=List_of_web_browsers&oldid=75212914, Accessed 2006 Sept 12.

[10] ANON. SOAP, 2006. http://en.wikipedia.org/w/index.php?title=SOAP&oldid=75067885, Accessed 2006 Sept 13.

[11] APPLE COMPUTER. Tunneling RTSP and RTP over HTTP, 2006. http://developer.apple.com/documentation/QuickTime/QTSS/Concepts/chapter_2_section_14.html, Accessed 2006 Sept 13.

[12] ATHANASIADES, N., ABLER, R., LEVINE, J., OWEN, H., AND 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, 2003), IEEE Computer Society, p. 63.

[13] AXELSSON, S. Research in intrusion-detection systems: A survey. Tech. Rep. 98-17, Department of Computer Engineering, Chalmers University of Technology, SE-412 96 Göteborg, Sweden, Dec. 1998.

[14] AXELSSON, S. The base-rate fallacy and its implications for the difficulty of intrusion detection. In *ACM Conference on Computer and Communications Security* (1999), pp. 1–7. http://www.ce.chalmers.se/staff/sax/difficulty.ps Accessed 19 August 2002.

[15] AXELSSON, S. On a difficulty of intrusion detection. In *Recent Advances in Intrusion Detection* (1999).

[16] AXELSSON, S. The base-rate fallacy and the difficulty of intrusion detection. *ACM Transactions on Information and Systems Security 3*, 3 (August 2000), 186–205.

[17] AXELSSON, S. Intrusion detection systems: A survey and taxonomy. Tech. Rep. 99-15, Dept. of Computer Engineering, Chalmers University of Technology, SE-412 96 Gteborg, Sweden, March 2000. http://www.ce.chalmers.se/staff/sax/taxonomy.ps Accessed 27 Feb 2002.

[18] BAI, Y., AND KOBAYASHI, H. Intrusion detection system: Technology and development. In *AINA '03: Proceedings of the 17th International Conference on Advanced Information Networking and Applications* (Washington, DC, USA, 2003), IEEE Computer Society, p. 710.

[19] BALTHROP, J., FORREST, S., AND GLICKMAN, M. Revisiting LISYS: Parameters and normal behavior. In *Proceedings of the 2002 Congress on Evolutionary Computation* (2002). http://www.cs.unm.edu/~forrest/publications/rev_lysis.ps Accessed 19 August 2002.

[20] BARHAM, P., DRAGOVIC, B., FRASER, K., HAND, S., HARRIS, T., HO, A., NEUGEBAUER, R., PRATT, I., AND WARFIELD, A. Xen and the art of virtualization. In *SOSP '03: Proceedings of the nineteenth ACM symposium on Operating systems principles* (New York, NY, USA, 2003), ACM Press, pp. 164–177.

[21] BASS, T. Multisensor data fusion for next generation distributed intrusion detection systems. In *Proceedings, 1999 IRIS National Symposium on Sensor and Data Fusion, May 1999.* (1999).

[22] BASS, T. Intrusion detection systems and multisensor data fusion. *Communications of the ACM 43*, 4 (April 2000), 99–105.

[23] BASS, T., AND GRUBER, D. A glimpse into the future of ID. *;login:* (September 1999). `http://www.usenix.org/publications/login/1999-9/features/future.html`. Accessed 29 July 2002.

[24] BERGHEL, H. The Code Red Worm. *Communications of the ACM 44*, 12 (Dec. 2001), 15–19.

[25] BHARGAVAN, K., CHANDRA, S., MCCANN, P. J., AND GUNTER, C. A. What packets may come: automata for network monitoring. *ACM SIGPLAN Notices 36*, 3 (2001), 206–219.

[26] BILAR, D., AND BURROUGHS, D. Introduction to state-of-the-art intrusion detection technologies. *Proceedings of the SPIE—The International Society for Optical Engineering 4232* (2001), 123–33.

[27] BLOEDORN, E., HILL, B., CHRISTIANSEN, A., SKORUPKA, C., TALBOT, L., AND TIVEL, J. Data mining for improving intrusion detection, 2000. `http://www.mitre.org/work/tech_papers/tech_papers_00/bloedorn_datamining/index.html` Accessed 26 June 2006.

[28] BONDY, J. A., AND MURTY, U. S. R. *Graph Theory with Applications*. North-Holland, 1976.

[29] BORISOV, N., GOLDBERG, I., AND WAGNER, D. Intercepting mobile communications: The insecurity of 802.11. In *Proceedings of the Seventh International Conference on Mobile Computing and Networking July 16-21, 2001, Rome, Italy* (2001).

[30] Ibm outsourced solution, 1998. `http://www.infoworld.com/cgi-bin/displayTC.pl?/980504sb3-ibm.htm` Accessed 26 June 2006.

[31] BRONSTEIN, A., DAS, J., DURO, M., FRIEDRICH, R., KLEYNER, G., MUELLER, M., SINGHAL, S., AND COHEN, I. Self-aware services: using

Bayesian networks for detecting anomalies in internet-based services. In *2001 International Symposium on Integrated Network Management, 14–18 May 2001, Seattle, WA, USA* (Piscataway, NJ, USA, 2001), G. Pavlou, N. Anerousis, and A. Liotta, Eds., IEEE, pp. 623–38.

[32] BRONSTEIN, A., DAS, J., DURO, M., FRIEDRICH, R., KLEYNER, G., MUELLER, M., SINGHAL, S., AND COHEN, I. Self-aware services: Using Bayesian networks for detecting anomalies in internet-based services. Tech. Rep. HPL-2001-23, Internet Storage and Systems Laboratory, HP Laboratories, Palo Alto, CA, US, Oct. 2001.

[33] BRUMLEY, D., LIU, L.-H., POOSANKAM, P., AND SONG, D. Design space and analysis of worm defense strategies. In *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security* (New York, NY, USA, 2006), ACM Press, pp. 125–137.

[34] BURGESS, M., HAUGERUD, H., STRAUMSNES, S., AND REITAN, T. Measuring system normality. *ACM Trans. Comput. Syst. 20*, 2 (2002), 125–160.

[35] CANNADY, J. Artificial neural networks for misuse detection. In *Proceedings of the 1998 National Information Systems Security Conference (NISSC'98) October 5–8 1998. Arlington, VA.* (1998), pp. 443–456.

[36] CERT COORDINATION CENTER. Unix configuration guidelines, June 2003. `http://www.cert.org/tech_tips/unix_configuration_guidelines.html` Accessed 8 February 2006.

[37] CERT COORDINATION CENTER. Windows NT configuration guidelines, April 2003. `http://www.cert.org/tech_tips/win_configuration_guidelines.html` Accessed 8 February 2006.

[38] CICCHELLO, O., AND KREMER, S. C. Inducing grammars from sparse data sets: a survey of algorithms and results. *J. Mach. Learn. Res. 4* (2003), 603–632.

[39] CLIFTON, C., AND GENGO, G. Developing custom intrusion detection filters using data mining. In *MILCOM 2000. 21st Century Military Communications Conference Proceedings* (2000).

[40] COHEN, C. F. CERT advisory CA-2002-17 Apache web server chunk handling vulnerability, July 2002. `http://www.cert.org/advisories/CA-2002-17.html`. Accessed 24 July 2002.

[41] COHEN, D. I. A. *Basic Techniques of Combinatorial Theory.* John Wiley & Sons, 1978.

[42] COHEN, W. W., AND SINGER, Y. Context-sensitive learning methods for text categorization. In *Proceedings of SIGIR-96, 19th ACM International Conference on Research and Development in Information Retrieval* (Zürich, CH, 1996), H.-P. Frei, D. Harman, P. Schäuble, and R. Wilkinson, Eds., ACM Press, New York, US, pp. 307–315.

[43] COMMUNITY, O. S. Community rules, November 2005. `http://www.snort.org/pub-bin/downloads.cgi`, accessed November 10, 2005.

[44] COMPUTER EMERGENCY RESPONSE TEAM (CERT). CERT advisory CA-2000-04 love letter worm, May 2000. `http://www.cert.org/advisories/CA-2000-04.html`.

[45] CORPORATION, M. Common vulnerabilities and exposures. `http://cve.mitre.org/` Accessed 16 June 2006.

[46] CROSBIE, M., DOLE, B., ELLIS, T., KRSUL, I., AND SPAFFORD, E. IDIOT—user guide. Tech. Rep. TR-96-050, Purdue University, West Lafayette, IN, US, Sept. 1996.

[47] CUPPENS, F. Managing alerts in a multi-intrusion detection environment. In *Seventeenth Annual Computer Security Applications Conference, 10-14 Dec. 2001, New Orleans, LA, USA* (Los Alamitos, CA, USA, 2001), IEEE Comput. Soc, pp. 22–31.

[48] CUPPENS, F., AND MIÉGE, A. Alert correlation in a cooperative intrusion detection framework. In *SP '02: Proceedings of the 2002 IEEE Symposium on Security and Privacy* (Washington, DC, USA, 2002), IEEE Computer Society, p. 202.

[49] CURRY, D., AND DEBAR, H. Intrusion detection message exchange format data model and extensible markup language (XML) document type definition, Dec. 2002. `http://www.ietf.org/internet-drafts/draft-ietf-idwg-idmef-xml-09.txt` Accessed 1 January 2003.

[50] CVE.MITRE.ORG. CVE-1999-0107, July 1999. `http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-0107` Accessed 3 September 2006.

[51] CVE.MITRE.ORG. CVE-2000-0097, September 2000. `http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2000-0097` Accessed 13 September 2006.

[52] CVE.MITRE.ORG. CVE-2001-0731, June 2002. `http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2001-0731` Accessed 1 October 2006.

[53] CVE.MITRE.ORG. CVE-2002-0392, April 2003. `http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2002-0392` Accessed 7 February 2006.

[54] CVE.MITRE.ORG. CVE-1999-1199, September 2004. `http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-1999-1199` Accessed 30 October 2005.

[55] CVE.MITRE.ORG. Cve-2006-0797, February 2006. `http://www.cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2006-0797` Accessed 21 June 2006.

[56] DAIN, O. M., AND CUNNINGHAM, R. K. Building scenarios from a heterogeneous alert stream. *IEEE Transactions on Systems, Man and Cybernetics* (2002). `http://www.ll.mit.edu/IST/pubs/2002_Building.pdf` Accessed 2006 June 26.

[57] DAMASHEK, M. Gauging similarity with n-grams: language-independent categorization of text. *Science 267*, 5199 (1995), 843–848.

[58] DANYLIW, R., DOUGHERTY, C., HOUSEHOLDER, A., AND RUEFLE, R. CERT advisory CA-2001-26 Nimda worm, September 2001. `http://www.cert.org/advisories/CA-2001-26.html`.

[59] DARLING, T., AND SHAYMAN, M. A Markov decision model for intruder location in IP networks. In *39th IEEE Conference on Decision and Control, 12–15 Dec. 2000, Sydney, NSW, Australia* (Piscataway, NJ, USA, 2000), vol. 2, IEEE, pp. 1858–63.

[60] DASGUPTA, D. Immunity-based intrusion detection system: a general framework. In *22nd National Information System Security Conference, 18–21 Oct. 1999, Arlington, VA, USA* (Gaithersburg, MD, USA, 1999), vol. 1, NIST, pp. 147–60.

[61] DAVIS, R. Using an expert system to peel the computer virus onion. *EDPACS 20*, 2 (August 1992), 1–12.

[62] DEBAR, H., BECKER, M., AND SIBONI, D. A neural network component for an intrusion detection system. In *1992 IEEE Computer Society Symposium on Research in Security and Privacy, 4-6 May 1992, Oakland, CA, USA* (1992), Los Alamitos, CA, USA : IEEE Computer Society Press, 1992, pp. 240–50.

[63] DEBAR, H., DACIER, M., AND WESPI, A. A revised taxonomy for intrusion-detection systems. Tech. Rep. RZ 3176 (# 93222), IBM Research, Zurich Research Laboratory, 8803 Rüschlikon, Switzerland, Oct. 1999.

[64] DEBAR, H., DACIER, M., AND WESPI, A. Towards a taxonomy of intrusion-detection systems. *Computer Networks 31*, 8 (Apr. 1999), 805–822.

[65] DEBAR, H., DACIER, M., AND WESPI, A. A revised taxonomy for intrusion-detection systems. *Annales des Telecommunications 55*, 7-8 (July-August 2000), 361–78.

[66] DEBAR, H., DACIER, M., WESPI, A., AND LAMPART, S. An experimentation workbench for intrusion detection systems. Tech. Rep. RZ 6519, IBM Research Division, Zurich Research Laboratory, 8803 Rüuschlikon, Switzerland, Sept. 1998.

[67] DENNING, D. An intrusion-detection model. In *1986 IEEE Symposium on Security and Privacy, 7-9 April 1986, Oakland, CA, USA* (1986), Washington, DC, USA : IEEE Comput. Soc. Press, 1986, pp. 118–31.

[68] DENNING, D. An intrusion-detection model. *IEEE Transactions on Software Engineering SE-13*, 2 (February 1987), 222–32.

[69] DEPARTMENT OF HEALTH AND HUMAN SERVICES. Health insurance portability and accountability act of 1996. 45 CFR parts 160 and 164, subparts A and E. Available online at `http://www.hhs.gov/ocr/hipaa`. Accessed 16 June 2006.

[70] D'HAESELEER, P. An immunological approach to change detection: theoretical results. In *9th IEEE Computer Security Foundations Workshop, 10-12 June 1996, Kenmare, Ireland* (Los Alamitos, CA, USA, 1996), IEEE Computer Society Press, pp. 18–26.

[71] D'HAESELEER, P., FORREST, S., AND HELMAN, P. An immunological approach to change detection: Algorithms, analysis and implications. In *1996 IEEE Symposium on Security and Privacy, 6-8 May 1996, Oakland, CA, USA* (1996), Los Alamitos, CA, USA : IEEE Computer Society Press, 1996, pp. 110–119. `http://cs.unm.edu/~forrest/publications/ieee-sp-96-neg-selec.pdf` Accessed 6 June 2002.

[72] DOWELL, C., AND RAMSTEDT, P. The ComputerWatch data reduction tool. In *13th National Computer Security Conference. Proceedings. Information Systems Security. Standards - the Key to the Future, 1-4 Oct. 1990, Washington, DC, USA* (Gaithersburg, MD, USA, 1990), NIST, pp. 99–108 vol.1.

[73] DOYLE, J., SHROBE, H., AND SZOLOVITS, P. On widening the scope of attack recognition languages. `http://medg.lcs.mit.edu/doyle/publications/` Accessed 29 July 2002, 2000.

[74] DRAELOS, T., COLLINS, M., DUGGAN, D., THOMAS, E., AND WUNSCH, D. Experiments on adaptive techniques for host-based intrusion detection. Tech. Rep. SAND2001-3065, Sandia National Laboratories, Albuquerque, NM, 2001. `http://infoserve.sandia.gov/cgi-bin/techlib/access-control.pl/2001/013065.pdf`. Accessed 12 December 2002.

[75] DUMOUCHEL, W. Computer intrusion detection based on Bayes factors for comparing command transition probabilities. Tech. Rep. TR91, National Institute of Statistical Sciences (NISS), 1999. `http://www.niss.org/technicalreports/tr91.pdf`. Accessed 29 July 2002.

[76] DUMOUCHEL, W., AND SCHONLAU, M. A comparison of test statistics for computer intrusion detection based on principal components regression of transition probabilities. In *Proceedings of the 30th Symposium on the Interface: Computing Science and Statistics* (1999), pp. 404–413.

[77] DURST, R., CHAMPION, T., WITTEN, B., MILLER, E., AND SPAGNUOLO, L. Testing and evaluating computer intrusion detection systems. *Communications of the ACM 42*, 7 (July 1999), 53–61.

[78] EASTLAKE, D., KHARE, R., AND MILLER, J. Selecting payment mechanisms over HTTP, 2006. `http://www.w3.org/TR/WD-jepi-uppflow-970106`, Accessed 2006 Sept 13.

[79] ECKMANN, S., VIGNA, G., AND KEMMERER, R. STATL: an attack language for state-based intrusion detection. *Journal of Computer Security 10*, 1–2 (2002), 71–103.

[80] EEYE DIGITAL SECURITY. SecureIIS^{TM}web server protection, 2004. At `http://www.eeye.com/html/Products/SecureIIS/index.html`. Accessed 17 March 2005.

[81] EICHMAN, K. Re: Possible codered connection attempts, July 20 2001. `http://lists.jammed.com/incidents/2001/07/0159.html` Accessed 12 January 2005.

[82] ENDLER, D. Intrusion detection. applying machine learning to Solaris audit data. In *14th Annual Computer Security Applications Conference, 7–11 Dec. 1998, Phoenix, AZ, USA* (Los Alamitos, CA, USA, 1998), IEEE Computer Society, pp. 268–79.

[83] ESKIN, E. Anomaly detection over noisy data using learned probability distributions. In *Proc. 17th International Conf. on Machine Learning* (2000), Morgan Kaufmann, San Francisco, CA, pp. 255–262.

[84] ESKIN, E., MILLER, M., ZHONG, Z., YI, G., LEE, W., AND STOLFO, S. Adaptive model generation for intrusion detection. In *Proceedings of the ACMCCS Workshop on Intrusion Detection and Prevention, Athens, Greece, 2000.* (2000).

[85] ESPONDA, F., FORREST, S., AND HELMAN, P. A formal framework for positive and negative detection schemes. *IEEE transactions on systems, man and cybernetics. Part B. Cybernetics* (2003). in press.

[86] ESTÉVEZ-TAPIADOR, J. M., GARCÍA-TEODORO, P., AND DÍAZ-VERDEJO, J. E. Measuring normality in http traffic for anomaly-based intrusion detection. *Journal of Computer Networks 45*, 2 (2004), 175–193.

[87] FALOUTSOS, C., AND OARD, D. W. A survey of information retrieval and filtering methods. Tech. Rep. CS-TR-3514, Electrical Engineering Department, University of Maryland, College Park, MD 20742, 1995.

[88] FIELDING, R., GETTYS, J., MOGUL, J., FRYSTYK, H., MASINTER, L., LEACH, P., AND BERNERS-LEE, T. Hypertext transfer protocol—HTTP/1.1, June 1999. RFC 2616. `ftp://ftp.isi.edu/in-notes/rfc2616.txt` Accessed 2002 Oct 2.

[89] FIXER. Default password list, June 2006. `http://www.phenoelit.de/dpl/dpl.html` Accessed 21 June 2006.

[90] FLAJOLET, P., GUIVARC'H, Y., SZPANKOWSKI, W., AND VALLE, B. Hidden pattern statistics. In *Automata, Languages and Programming. 28th International Colloquium, ICALP 2001. Proceedings, 8–12 July 2001, Crete, Greece* (Berlin, Germany, 2001), Springer-Verlag, pp. 152–65.

[91] FORREST, S., HOFMEYR, S., SOMAYAJI, A., AND LONGSTAFF, T. A sense of self for Unix processes. In *1996 IEEE Symposium on Security and Privacy, 6-8 May 1996, Oakland, CA, USA* (Los Alamitos, CA, USA, 1996), IEEE Computer Society Press, pp. 120–128.

[92] FORREST, S., PERELSON, A. S., ALLEN, L., AND CHERUKURI, R. Self-nonself discrimination in a computer. In *1994 IEEE Computer Society Symposium on Research in Security and Privacy, 16-18 May 1994, Oakland, CA, USA* (1994), Los Almitos, CA, USA : IEEE Computer Society Press, 1994, pp. 202–212.

[93] FYODOR, Y. 'SnortNet'—a distributed intrusion detection system, June 2000. `http://snortnet.scorpions.net/snortnet.ps` Accessed 5 August 2002.

[94] GHOSH, A., WANKEN, J., AND CHARRON, F. Detecting anomalous and unknown intrusions against programs. In *Proceedings of the 1998 Annual Computer Security*

Applications Conference (ACSAC'98), December 1998. (Los Alamitos, CA, USA, 1998), IEEE Computer Society, pp. 259–267.

[95] GHOSH, A. K., SCHWARTZBARD, A., AND SCHATZ, M. Learning program behavior profiles for intrusion detection. In *Proceedings 1st USENIX Workshop on Intrusion Detection and Network Monitoring* (Apr. 1999), pp. 51–62.

[96] GOAN, T. A cop on the beat: collecting and appraising intrusion. *Communications of the ACM 42*, 7 (July 1999), 46–52.

[97] GOLAND, Y., WHITEHEAD, E., FAIZI, A., CARTER, S., AND JENSEN, D. Http extensions for distributed authoring—webdav, February 1999. RFC 2518. `ftp://ftp.rfc-editor.org/in-notes/rfc2518.txt`. Accessed 23 February 2005.

[98] GOLD, E. M. Language identification in the limit. *Information and Control 10* (1967), 447–474.

[99] GRAFF, M. G., AND VAN WYK, K. R. *Secure Coding: Principles and Practices.* O'Reilly and Associates, Sebastopol, CA, 2003.

[100] GU, G., FOGLA, P., DAGON, D., LEE, W., AND SKORIĆ, B. Measuring intrusion detection capability: an information-theoretic approach. In *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security* (New York, NY, USA, 2006), ACM Press, pp. 90–101.

[101] HAINES, J., ROSSEY, L., LIPPMANN, R., AND CUNNINGHAM, R. Extending the DARPA off-line intrusion detection evaluations. In *DARPA Information Survivability Conference and Exposition II. DISCEX'01, 12–14 June 2001, Anaheim, CA, USA* (Los Alamitos, CA, USA, 2001), vol. 1, IEEE Computer Society, pp. 35–45.

[102] HAINES, J. W., LIPPMANN, R. P., FRIED, D. J., TRAN, E., BOSWELL, S., AND ZISSMAN, M. A. 1999 DARPA intrusion detection system evaluation: Design and procedures. Tech. Rep. TR-1062, Lincoln Laboratory, Massachusetts Institute of Technology, Lexington, MA, USA, Feb. 2001.

[103] HANCOCK, J., AND WINTZ, P. *Signal Detection Theory.* McGraw-Hill, 1966.

[104] HARMER, P., WILLIAMS, P., GUNSCH, G., AND LAMONT, G. An artificial immune system architecture for computer security applications. *IEEE Transactions on Evolutionary Computation 6*, 3 (June 2002), 252–80.

[105] HATCH, B., LEE, J., AND KURTZ, G. *Hacking Linux Exposed: Lunix Security Secrets & Solutions.* Osborne/McGraww-Hill, 2001.

[106] HEBERLEIN, L. Network security monitor (NSM)—final report. Tech. rep., University of California at Davis Computer Security Lab, 1995. Lawrence Livermore National Laboratory project deliverable. `http://seclab.cs.ucdavis.edu/papers/NSM-final.pdf`.

[107] HEBERLEIN, L., DIAS, G., LEVITT, K., MUKHERJEE, B., WOOD, J., AND WOLBER, D. A network security monitor. In *1990 IEEE Computer Society Symposium on Research in Security and Privacy, 7–9 May 1990, Oakland, CA, USA* (Los Alamitos, CA, USA, 1990), IEEE Computer Society Press, pp. 296–304.

[108] HEBERLEIN, L. T., DIAS, G. V., LEVITT, K. N., MUKHERJEE, B., WOOD, J., AND WOLBER, D. A network security monitor. In *Proceedings of the IEEE Symposium on Security and Privacy* (1990), IEEE Press, pp. 296–304.

[109] HELMAN, P., AND LIEPINS, G. Statistical foundations of audit trail analysis for the detection of computer misuse. *IEEE Transactions on Software Engineering 19*, 9 (September 1993), 886–901.

[110] HELMER, G., WONG, J., HONAVAR, V., AND MILLER, L. Automated discovery of concise predictive rules for intrusion detection. *Journal of Systems and Software 60*, 3 (February 2002), 165–75.

[111] HOCHBERG, J., JACKSON, K., STALLINGS, C., MCCLARY, J., DUBOIS, D., AND FORD, J. Nadir: an automated system for detecting network intrusion and misuse. *Computers & Security 12*, 3 (May 1993), 235–48.

[112] HOFMEYR, S., AND FORREST, S. Immunity by design: an artificial immune system. In *Proceedigns GECCO-99. Genetic and Evolutionary Computation Conference. Eighth International Conference on Genetic Algorithms (ICGA-99) and the Fourth Annual Gentic Programming Conference (GP-99), 13-17 July 1999, Orlando, FL, USA* (San Francisco, CA, USA, 1999), W. Banzhaf, J. Daida, A. Eiben, M. Garzon, V. Honavar, M. Jakiela, and R. Smith, Eds., Morgan Kaufmann Publishers, pp. 1289–96 vol.2.

[113] HOFMEYR, S. A. *An Immunological Model of Distributed Detection and Its Application to Computer Security*. PhD thesis, University of New Mexico, Computer Science Department, May 1999.

[114] HOFMEYR, S. A., AND FORREST., S. Immunizing computer networks: Getting all the machines in your network to fight the hacker disease. In *1999 IEEE Symposium on Security and Privacy* (1999), pp. 9–12.

[115] HOFMEYR, S. A., AND FORREST, S. Architecture for an artificial immune system. *Evolutionary Computation 7*, 1 (2000), 1289–1296.

[116] HOFMEYR, S. A., FORREST, S., AND SOMAYAJI, A. Intrusion detection using sequences of system calls. *Journal of Computer Security 6*, 3 (1998), 151–80. `http://cs.unm.edu/~forrest/publications/int_decssc.pdf` Accessed 13 March 2002.

[117] HOGLUND, G., AND MCGRAW, G. *Exploiting Software: How to Break Code*. Addison Wesley, Reading, MA, 2004.

[118] ILGUN, K. USTAT: A real-time intrusion detection system for UNIX. Master's thesis, University of California Santa Barbara, Nov. 1992.

[119] ILGUN, K. USTAT: a real-time intrusion detection system for UNIX. In *IEEE Symposium on Research in Security and Privacy, 24-26 May 1993, Oakland, CA, USA* (1993), Los Alamitos, CA, USA : IEEE Comput. Soc. Press, 1993, pp. 16–28.

[120] ILGUN, K., KEMMERER, R., AND PORRAS, P. State transition analysis: a rule-based intrusion detection approach. *IEEE Transactions on Software Engineering 21*, 3 (March 1995), 181–99.

[121] INGHAM, K., AND FORREST, S. A history and survey of network firewalls. Tech. Rep. TR-CS-2002-37, University of New Mexico Computer Science Department, 2002. `http://www.cs.unm.edu/colloq-bin/tech_reports.cgi?ID=TR-CS-2002-37`. Accessed 29 December 2002.

[122] INGHAM, K., AND FORREST, S. Network firewalls. In *Enhancing Computer Security with Smart Technology* (2006), V. R. Vemuri, Ed., Auerbach Publications, pp. 9–40. Chapter 2.

[123] ISO/TC97/SC16. Reference model of open systems interconnection. Tech. Rep. N. 227, International Organization for Standardization, June 1979.

[124] JACKSON, K. A. Intrusion detection system (IDS) product survey. Tech. Rep. LA-UR-99-3883, Distributed Knowledge Systems Team, Computer Research and Applications Group, Computing, Information, and Communications Division, Los Alamos National Laboratory, Los Alamos, New Mexico, USA, 1999.

[125] JACKSON, K. A., DUBOIS, D. H., AND STALLINGS, C. A. NADIR–A prototype network intrusion detection system. Tech. Rep. LA-UR–90-3726, Los Alamos National Laboratory, Los Alamos, NM USA 87545, 1990.

[126] JAVITZ, H., AND VALDES, A. The SRI IDES statistical anomaly detector. In *1991 IEEE Computer Society Symposium on Research in Security and Privacy, 20–22 May 1991, Oakland, CA, USA* (Los Alamitos, CA, USA, 1991), IEEE Computer Society Press, pp. 316–326.

[127] JHA, S., TAN, K., AND MAXION, R. Markov chains classifiers and intrusion detection. In *14th IEEE Computer Security Foundations Workshop, 11–13 June 2001, Cape Breton, NS, Canada* (Los Alamitos, CA, USA, 2001), IEEE Computer Society, pp. 206–19.

[128] JONES, A., AND LI, S. Temporal signatures for intrusion detection. In *Seventeenth Annual Computer Security Applications Conference, 10–14 Dec. 2001, New Orleans, LA, USA* (Los Alamitos, CA, USA, 2001), IEEE Computer Society, pp. 252–61.

[129] JONES, A. K., AND LIN, Y. Application intrusion detection using language library calls. In *Seventeenth Annual Computer Security Applications Conference, 10-14 Dec. 2001, New Orleans, LA, USA* (Los Alamitos, CA, USA, 2001), IEEE Computer Society, pp. 22–31.

[130] JONES, A. K., AND SIELKEN, R. S. Computer system intrusion detection: A survey. Tech. rep., University of Virginia Computer Science Department, 1999. `http://www.cs.virginia.edu/~jones/IDS-research/Documents/jones-sielken-survey-v11.pdf` Accessed 6 June 2002.

[131] JULISCH, K. Mining alarm clusters to improve alarm handling efficiency. In *Seventeenth Annual Computer Security Applications Conference, 10-14 Dec. 2001, New Orleans, LA, USA* (Los Alamitos, CA, USA, 2001), IEEE Comput. Soc, pp. 12–21.

[132] JULISCH, K. Clustering intrusion detection alarms to support root cause analysis. *ACM Trans. Inf. Syst. Secur. 6*, 4 (2003), 443–471.

[133] KAN KVARNSTRÖM, H. A survey of commercial tools for intrusion detection. Tech. Rep. 99-8, Department of Computer Engineering, Chalmers University of Technology, Götenborg, Sweden, Oct. 1999.

[134] KANSON, P. H. All public hospitals in Gothenburg Sweden crippled by Nimda. *Forum on Risks to the Public in Computers and Related Systems 21*, 67 (Oct. 2001). `http://catless.ncl.ac.uk/Risks/21.67.html#subj13`. Accessed 27 Dec 2002.

[135] KEMMERER, R., AND VIGNA, G. Intrusion detection: a brief history and overview. *Computer 35*, 4 (Apr. 2002), 27–30.

[136] KIM, J. An artificial immune system for network intrusion detection. In *Graduate Student Workshop, Genetic and Evolutionary Computation Conference, Orlando, Florida. July 13–17 (GECCO-99)* (July 1999), U.-M. O'Reilly, Ed., pp. 369–370.

[137] KIM, J., AND BENTLEY, P. The artificial immune model for network intrusion detection. In *7th European Conference on Intelligent Techniques and Soft Computing (EUFIT'99), Aachen, Germany* (1999).

[138] Kim, J., and Bentley, P. The human immune system and network intrusion detection. In *7th European Congress on Intelligent Techniques and Soft Computing (EUFIT '99), Aachen, Germany, September 13–19* (1999).

[139] Kim, J., and Bentley, P. Negative selection and niching by an artificial immune system for network intrusion detection. In *Late Breaking Papers at the 1999 Genetic and Evolutionary Computation Conference* (1999), pp. 149–158.

[140] Kim, J., and Bentley, P. J. Towards an artificial immune system for network intrusion detection: An investigation of dynamic clonal selection. In *Proceedings of the Congress on Evolutionary Computation (CEC-2002), Honolulu, HI* (May 2002), pp. 1015–1020.

[141] Kiselyov, O. A network file system over HTTP: remote access and modification of files and files. In *1999 USENIX Annual Technical Conference, FREENIX Track* (Berkeley, CA, 1999), Usenix Association. `http://www.usenix.org/publications/library/proceedings/usenix99/kiselyov.html`, Accessed 2006 Sept 13.

[142] Ko, C., Fink, G., and Levitt, K. Automated detection of vulnerabilities in privileged programs by execution monitoring. In *Tenth Annual Computer Security Applications Conference, 5-9 Dec. 1994, Orlando, FL, USA* (Los Alamitos, CA, USA, 1994), IEEE Computer Society Press, pp. 134–144.

[143] Kohout, L., Yasinsac, A., and McDuffie, E. Activity profiles for intrusion detection. In *2002 Annual Meeting of the North American Fuzzy Information Processing Society Proceedings, 27-29 June 2002, New Orleans, LA, USA* (Piscataway, NJ, USA, 2002), J. Keller and O. Nasraoui, Eds., IEEE, pp. 463–8.

[144] Kosoresow, A. P., and Hofmeyr, S. A. Intrusion detection via system call traces. *IEEE Software 14*, 5 (Sep–Oct 1997), 35–42.

[145] Kruegel, C., Mutz, D., Valeur, F., and Vigna, G. On the detection of anomalous system call arguments. In *ESORICS 2003: 8th European Symposium on Research in Computer Security*, vol. 2808 of *Lecture Notes in Computer Science*. Springer, 2003, pp. 326–343.

[146] Kruegel, C., and Vigna, G. Anomaly detection of web-based attacks. In *Proceedings of the 10th ACM conference on Computer and communications security* (2003), ACM Press, pp. 251–261.

[147] Kruegel, C., Vigna, G., and Robertson, W. A multi-model approach to the detection of web-based attacks. *Comput. Networks 48*, 5 (2005), 717–738.

[148] KUMAR, S. *Classification and detection of computer intrusions*. PhD thesis, Purdue University, 1995.

[149] KUMAR, S., AND SPAFFORD, E. H. A Pattern Matching Model for Misuse Intrusion Detection. In *Proceedings of the 17th National Computer Security Conference* (1994), pp. 11–21.

[150] KUMAR, S., AND SPAFFORD, E. H. An Application of Pattern Matching in Intrusion Detection. Tech. Rep. 94–013, Department of Computer Sciences, Purdue University, 1994.

[151] KUMAR, S., AND SPAFFORD, E. H. A software architecture to support misuse intrusion detection. In *Proceedings of the 18th National Information Security Conference* (1995), pp. 194–204.

[152] LANE, T., AND BRODLEY, C. Approaches to online learning and concept drift for user identification in computer security. In *Fourth International Conference on Knowledge Discovery and Data Mining, 27–31 Aug. 1998, New York, NY, USA* (Menlo Park, CA, USA, 1998), P. Agrawal, R.; Stolorz, Ed., AAAI Press, pp. 259–63.

[153] LANE, T., AND BRODLEY, C. Temporal sequence learning and data reduction for anomaly detection. In *5th ACM Conference on Computer and Communications Security, 2–5 Nov. 1998, San Francisco, CA, USA* (New York, NY, USA, 1998), ACM, pp. 150–8.

[154] LANE, T., AND BRODLEY, C. E. An application of machine learning to anomaly detection. In *20th Annual National Information Systems Security Conference* (1997), vol. 1, pp. 366–380.

[155] LANE, T., AND BRODLEY, C. E. Detecting the abnormal: Machine learning in computer security. Tech. Rep. ECE-97-1, Department of Electrical and Computer Engineering, Purdue University, West Lafayette, IN 47907, Jan. 1997.

[156] LANE, T., AND BRODLEY, C. E. Sequence matching and learning in anomaly detection for computer security. In *Proceedings of the AAAI-97 Workshop on AI Approaches to Fraud Detection and Risk Management* (1997), pp. 43–49.

[157] LANE, T. D. *Machine learning techniques for the domain of anomaly detection for computer security*. PhD thesis, Department of Electrical and Computer Engineering, Purdue University, July 1998.

[158] LANG, K. J. Random DFA's can be approximately learned from sparse uniform examples. In *Proceedings of the Fifth ACM Workshop on Computational Learning Theory* (New York, N.Y., 1992), ACM, pp. 45–52.

[159] LANG, K. J., PEARLMUTTER, B. A., AND PRICE, R. A. Results of the Abbadingo One DFA learning competition and a new evidence-driven state merging algorithm. *Lecture Notes in Computer Science 1433* (1998). Proceedings of ICGI-98.

[160] LEE, S., AND HEINBUCH, D. Training a neural-network based intrusion detector to recognize novel attacks. *IEEE Transactions on Systems, Man & Cybernetics, Part A (Systems & Humans) 31*, 4 (July 2001), 294–9.

[161] LEE, W. *A Data Mining Framework for Constructing Features and Models for Intrusion Detection Systems*. PhD thesis, Columbia University, 1999.

[162] LEE, W., AND STOLFO, S. A framework for constructing features and models for intrusion detection systems. *ACM Transactions on Information and Systems Security 3*, 4 (November 2000), 227–61.

[163] LEE, W., STOLFO, S., AND MOK, K. A data mining framework for building intrusion detection models. In *1999 IEEE Symposium on Security and Privacy, 9–12 May 1999, Oakland, CA, USA* (Los Alamitos, CA, USA, 1999), IEEE Computer Society, pp. 120–32.

[164] LEE, W., AND STOLFO, S. J. Data mining approaches for intrusion detection. In *Proceedings of the 7th Usenix Security Symposium* (1998), Usenix Association.

[165] LEE, W., STOLFO, S. J., AND MOK, K. W. Mining in a data-flow environment: experience in network intrusion detection. In *Proceedings of the fifth ACM SIGKDD international conference on Knowledge discovery and data mining* (1999), ACM Press, pp. 114–124.

[166] LEE, W., AND XIANG, D. Information-theoretic measures for anomaly detection. In *2001 IEEE Symposium on Security and Privacy. S&P 2001, 14–16 May 2001, Oakland, CA, USA* (2001), pp. 130–143.

[167] LEUNG, K., AND LECKIE, C. Unsupervised anomaly detection in network intrusion detection using clusters. In *CRPIT '38: Proceedings of the Twenty-eighth Australasian conference on Computer Science* (Darlinghurst, Australia, Australia, 2005), Australian Computer Society, Inc., pp. 333–342.

[168] LI, Z., DAS, A., AND ZHOU, J. Model generalization and its implications on intrusion detection. In *Applied Cryptography and Network Security, Third International Conference, ACNS 2005, New York, NY, USA, June 7-10, 2005, Proceedings* (2005), pp. 222–237.

[169] LINDQVIST, U., AND PORRAS, P. Detecting computer and network misuse through the production-based expert system toolset (P-BEST). In *1999 IEEE Symposium on Security and Privacy, 9–12 May 1999, Oakland, CA, USA* (Los Alamitos, CA, USA, 1999), IEEE Computer Society, pp. 146–161.

[170] LINDQVIST, U., AND PORRAS, P. A. eXpert-BSM: A host-based intrusion detection solution for Sun Solaris. In *Proceedings of the 17th Annual Computer Security Applications Conference (ACSAC 2001)* (New Orleans, Louisiana, Dec. 2001), IEEE Computer Society, pp. 240–251.

[171] LIPPMANN, R., FRIED, D., GRAF, I., HAINES, J., KENDALL, K., MCCLUNG, D., WEBER, D., WEBSTER, S., WYSCHOGROD, D., CUNNINGHAM, R., AND ZISSMAN, M. Evaluating intrusion detection systems: the 1998 DARPA off-line intrusion detection evaluation. In *DARPA Information Survivability Conference and Exposition, 2000. DISCEX '00. Proceedings, Volume: 2* (1999), pp. 12–26.

[172] LIPPMANN, R., HAINES, J., FRIED, D., KORBA, J., AND DAS, K. The 1999 DARPA off-line intrusion detection evaluation. *Computer Networks 34*, 4 (October 2000), 579–95.

[173] LIPPMANN, R., HAINES, J., FRIED, D., KORBA, J., AND DAS, K. Analysis and results of the 1999 DARPA off-line intrusion detection evaluation. In *Recent Advances in Intrusion Detection. Third International Workshop, RAID 2000, 2–4 Oct. 2000, Toulouse, France* (Berlin, Germany, 2000), H. Debar, L. Me, and S. Wu, Eds., Springer-Verlag, pp. 162–82.

[174] LITTMAN, M. L., AND ACKLEY, D. H. Adaptation in constant utility nonstationary environments. In *Proceedings of the Fourth International Conference on Genetic Algorithms* (San Mateo, CA, 1991), R. K. Belew and L. B. Booker, Eds., Morgan Kaufmann, pp. 136–142.

[175] LUNDIN, E., AND JONSSON, E. Some practical and fundamental problems with anomaly detection. In *Proceedings of the Fourth Nordic Workshop on Secure IT systems (NORDSEC'99)* (1999).

[176] LUNT, T. F. Automated audit trail analysis and intrusion detection: A survey. In *11th National Computer Security Conference* (Oct. 1988).

[177] LUNT, T. F. Detecting Intruders in Computer Systems. In *1993 Conference on Auditing and Computer Technology* (1993). `http://www.sdl.sri.com/papers/c/a/canada93/canada93.ps.gz` Accessed 22 August 2002.

[178] MAHONEY, M. V. Network traffic anomaly detection based on packet bytes. In *Proceedings of the 2003 ACM Symposium on Applied computing* (2003), ACM Press, pp. 346–350.

[179] MAHONEY, M. V., AND 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* (2002), ACM Press, pp. 376–385.

[180] MAHONEY, M. V., AND CHAN, P. K. Learning rules for anomaly detection of hostile network traffic. In *ICDM '03: Proceedings of the Third IEEE International Conference on Data Mining* (Washington, DC, USA, 2003), IEEE Computer Society, p. 601.

[181] MARCEAU, C. Characterizing the behavior of a program using multiple-length n-grams. In *New Security Paradigms Workshop 2000, 18–22 Sept. 2000, Ballycotton, Ireland* (New York, NY, USA, 2001), ACM, pp. 101–10. `http://www.atc-nycorp.com/papers/Marceau_multiLengthStrings.pdf` Accessed 13 August 2002.

[182] MAY, J., PETERSON, J., AND BAUMAN, J. Attack detection in large networks. In *DARPA Information Survivability Conference and Exposition II. DISCEX'01, 12–14 June 2001, Anaheim, CA, USA* (Los Alamitos, CA, USA, 2001), vol. 1, IEEE Computer Society, pp. 15–21.

[183] MAYFIELD, J., MCNAMEE, P., AND PIATKO, C. The JHU/APL HAIRCUT system at TREC-8. In *Information Technology: Eighth Text REtrieval Conference (TREC-8), 16-19 Nov. 1999, Gaithersburg, MD, USA* (2001), pp. 445–452.

[184] MCGRAW, G. *Software Security : Building Security In*. Addison-Wesley, 2006.

[185] MCHUGH, J. The 1998 Lincoln Laboratory IDS evaluation—a critique. In *Recent Advances in Intrusion Detection. Third International Workshop, RAID 2000, 2–4 Oct. 2000, Toulouse, France* (Berlin, Germany, 2000), H. Debar, L. Me, and S. Wu, Eds., Springer-Verlag, pp. 145–61.

[186] 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 (November 2000), 262–94.

[187] MCHUGH, J. Intrusion and intrusion detection. *International Journal of Information Security 1*, 1 (August 2001), 14–35.

[188] MICHAEL, C., AND GHOSH, A. Using finite automata to mine execution data for intrusion detection: a preliminary report. In *Recent Advances in Intrusion Detection. Third International Workshop, RAID 2000, 2-4 Oct. 2000, Toulouse, France* (Berlin, Germany, 2000), H. Debar, L. Me, and S. Wu, Eds., Springer-Verlag, pp. 66–79.

[189] MICHAEL, C., AND GHOSH, A. Simple state-based approaches to program-based anomaly detection. *ACM Transactions on Information and Systems Security 5*, 3 (August 2002), 203 – 37.

[190] MICHAEL, C. C., AND GHOSH, A. Two state-based approaches to program-based anomaly detection. In *Annual Computer Security Applications Conference: Practical Solutions to Real Security Problems December 11–15, 2000 New Orleans, LA, USA* (Dec. 2000), Applied Computer Security Associates, IEEE Computer Society, Los Alamitos, CA, USA, pp. 21–30. `http://www.acsac.org/2000/papers/96.pdf` Accessed 13 August 2002.

[191] MICHAEL, C. C., AND GHOSH, A. Simple, state-based approaches to program-based anomaly detection. *ACM Trans. Inf. Syst. Secur. 5*, 3 (2002), 203–237.

[192] MICROSOFT CORPORATION. Exchange server 2003 RPC over HTTP deployment scenarios, 2006. `http://www.microsoft.com/technet/prodtechnol/exchange/2003/library/ex2k3rpc.mspx`, Accessed 2006 Sept 13.

[193] MICROSOFT CORPORATION. Windows server update services frequently asked questions, 2006. `http://www.microsoft.com/windowsserversystem/updateservices/evaluation/faqs.mspx` Accessed 12 February 2006.

[194] MILLER, E. L., SHEN, D., LIU, J., AND NICHOLAS, C. Performance and scalability of a large-scale N-gram based information retrieval system. *Journal of Digital Information (online refereed journal)* (2000).

[195] MITCHELL, T. M. *Artificial Neural Networks.* WCB/McGraw-Hill, 1997, ch. 4, pp. 81–127.

[196] MOORE, D., PAXSON, V., SAVAGE, S., SHANNON, C., STANIFORD, S., AND WEAVER, N. Inside the slammer worm. *IEEE Security and Privacy 01*, 4 (2003), 33–39.

[197] MUKHERJEE, B., HEBERLEIN, L., AND LEVITT, K. Network intrusion detection. *IEEE Network 8*, 3 (May 1994), 26–41.

[198] MUKKAMALA, S., JANOSKI, G., AND SUNG, A. Intrusion detection using neural networks and support vector machines. In *2002 International Joint Conference on Neural Networks (IJCNN), 12–17 May 2002, Honolulu, HI, USA* (Piscataway, NJ, USA, 2002), vol. 2, IEEE, pp. 1702–1707. `http://www.cs.nmt.edu/~IT/papers/hawaii7.pdf` Accessed 13 August 2002.

[199] NETWORK ASSOCIATES, INC. Mcafee.com - virusscan home edition 7.0. `http://www.mcafee.com/myapps/vs7/` Accessed 1 January 2003.

[200] NEUMANN, P. G., AND PORRAS, P. A. Experience with EMERALD to date. In *First USENIX Workshop on Intrusion Detection and Network Monitoring (ID'99),*

9–12 April 1999, Santa Clara, CA, USA (apr 1999), USENIX Association, Berkeley, CA, USA, pp. 73–80. `http://www.sdl.sri.com/papers/det99/` Accessed 20 August 2002.

[201] NEWMAN, D., GIORGIS, T., AND YAVARI-ISSALOU, F. Intrusion detection systems: suspicious finds. *Data Communications International 27*, 11 (August 1998), 72–8, 80, 82. Online at `http://www.networkmagazine.com/article/printableArticle?doc_id=DCM20000510S0034` and `http://www.data.com/article/DCM20000510S0034`. Accessed 29 December 2002.

[202] NING, P., CUI, Y., AND REEVES, D. S. Constructing attack scenarios through correlation of intrusion alerts. In *Proceedings of the 9th ACM conference on Computer and communications security* (2002), ACM Press, pp. 245–254.

[203] NING, P., CUI, Y., REEVES, D. S., AND XU, D. Techniques and tools for analyzing intrusion alerts. *ACM Trans. Inf. Syst. Secur. 7*, 2 (2004), 274–318.

[204] NING, P., JAJODIA, S., AND WANG, X. S. Abstraction-based intrusion detection in distributed environments. *ACM Trans. Inf. Syst. Secur. 4*, 4 (2001), 407–452.

[205] NTOULAS, A., CHO, J., AND OLSTON, C. What's new on the web? the evolution of the web from a search engine perspective. In *13th International World Wide Web Conference* (New York, NY, USA, 2004), ACM Press.

[206] OLIVERIA, A. L., AND SILVA, J. Efficient search techniques for the inference of minimum sized finite automata. In *Proceedings of the Fifth String Processing and Information Retrieval Symposium* (1998), IEEE Computer Press, pp. 81–89.

[207] PATTON, S., YURCIK, W., AND DOSS, D. An Achilles' heel in signature-based IDS: Squealing false positives in SNORT. In *RAID 2001 Fourth International Symposium on Recent Advances in Intrusion Detection October 10–12, 2001, Davis, CA, USA* (2001). Available online only. `http://www.raid-symposium.org/raid2001/papers/patton_yurcik_doss_raid2001.pdf` Accessed 3 January 2003.

[208] PAULA, F. S., REIS, M. A., FERNANDES, D. M., AND GEUS, P. L. ADenoIdS: A hybrid IDS based on the immune system. In *Proceedings of the ICONIP2002: 9th International Conference on Neural Information Processing, Special Session on Artificial Immune Systems and Their Applications* (Nov. 2002). `http://www.dcc.unicamp.br/~ra000504/papers/AdenoidsMod_ICONIP2002.pdf` Accessed 25 April 2003.

[209] PAXSON, V. Bro: a system for detecting network intruders in real-time. In *Seventh USENIX Security Symposium, 26–29 Jan. 1998, San Antonio, TX, USA* (Berkeley, CA, USA, 1998), USENIX Association, pp. 31–51.

[210] PAXSON, V. Bro: a system for detecting network intruders in real-time. *Computer Networks 31*, 23–24 (December 1999), 2435–63.

[211] PHPMONKEY. Code Red - Neverside.com, August 7 2001. `http://forums.neverside.com/view/thread115/` Accessed 12 January 2005.

[212] PORRAS, P. A., AND NEUMANN, P. G. EMERALD: conceptual overview statement. `http://www.sdl.sri.com/papers/emerald-position1/` Accessed 20 August 2002., dec 1996.

[213] PORRAS, P. A., AND VALDES, A. Live traffic analysis of TCP/IP gateways. In *Internet Society's Networks and Distributed Systems Security Symposium* (March 1998). `http://www.sdl.sri.com/papers/gateway98/` Accessed 20 August 2002.

[214] PUKETZA, N., CHUNG, M., OLSSON, R., AND MUKHERJEE, B. A software platform for testing intrusion detection systems. *IEEE Software 14*, 5 (September 1997), 43–51.

[215] PUKETZA, N. J., ZHANG, K., CHUNG, M., MUKHERJEE, B., AND OLSSON, R. A. A methodology for testing intrusion detection systems. *IEEE Transactions on Software Engineering 22*, 10 (1996), 719–729.

[216] RAE, D., AND LUDLOW, D. Halt! who goes there? [internet intrusion detection benchtest]. *Network News (UK Edition)* (February 2000), 31–7.

[217] REIS, M. A., PAULA, F. S., FERNANDES, D. M., AND GEUS, P. L. A hybrid ids architecture based on the immune system. In *Anais do Wseg2002: Workshop em Seguranc, a de Sistemas Computacionais, Búzios, RJ, Maio de 2002. Workshop realizado durante o SBRC2002: Simpósio Brasileiro de Redes de Computadores* (May 2002). `http://www.dcc.unicamp.br/~ra000504/papers/IDShyb_WSEG2002.pdf` Accessed 25 April 2003.

[218] ROBERTSON, W., VIGNA, G., KRUEGEL, C., AND 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* (2006), Internet Society. `http://www.isoc.org/isoc/conferences/ndss/06/proceedings/html/2006/papers/anomaly_signatures.pdf` Accessed 12 February 2006.

[219] ROBINSON, D., AND COAR, K. The common gateway interface (cgi) version 1.1, October 2004. RFC 3875. `ftp://ftp.rfc-editor.org/in-notes/rfc3875.txt` Accessed 2006 Sept 12.

[220] ROESCH, M. Snort—lightweight intrusion detection for networks. In *13th Systems Administration Conference—LISA '99* (1999), pp. 229–238. `http://www.usenix.org/events/lisa99/roesch.html` Accessed 30 June 2002.

[221] ROOT-NINJA TAK. Re: Under attack...?, August 11 2003. `http://www.shroomery.org/forums/showflat.php/Number/1802533` Accessed 12 January 2005.

[222] RYAN, J., LIN, M.-J., AND MIIKKULAINEN, R. Intrusion detection with neural networks. In *Advances in Neural Information Processing Systems* (1998), M. I. Jordan, M. J. Kearns, and S. A. Solla, Eds., vol. 10, The MIT Press.

[223] SALGANICOFF, M. Density-adaptive learning and forgetting. In *International Conference on Machine Learning* (1993), pp. 276–283.

[224] SCOTT, S. L. Detecting network intrusion using a markov modulated nonhomogeneous poisson process. *Journal of the American Statistical Association* (2002). Submitted, under revision. Available at `http://www-rcf.usc.edu/~sls/mmnhpp.ps` Accessed 31 March 2003.

[225] SCOTT, S. L., AND SMYTH, P. The markov modulated poisson process and markov poisson cascade with applications to web traffic modeling. *Bayesian Statistics 7* (2003).

[226] SECURITIES AND EXCHANGE COMMISSION. Sarbanes-oxley act of 2002. 17 CFR Parts 240, 249, 270 and 274.

[227] SECURITYFOCUS. What is bugtraq? `http://online.securityfocus.com/popups/forums/bugtraq/intro.shtml` Accessed 10 January 2003.

[228] SECURITYFOCUS.COM. Multiple vendor "out of band" data (winnuke.c) DoS vulnerability, May 1997. Vulnerability database. `http://www.securityfocus.com/bid/2010` Accessed 2002 Feb 20.

[229] SECURITYFOCUS.COM. Microsoft internet explorer mhtml uri buffer overflow vulnerability, June 2006. SecurityFocus Vulnerability Database entry 18198. `http://www.securityfocus.com/bid/18198` Accessed 2006 June 21.

[230] SEKAR, R., BENDRE, M., DHURJATI, D., AND BOLLINENI, P. A fast automaton-based method for detecting anomalous program behaviors. In *IEEE Symposium on Security and Privacy* (2001), IEEE, pp. 144–155.

[231] SEKAR, R., BOWEN, T., AND SEGAL, M. On preventing intrusions by process behavior monitoring. In *Workshop on Intrusion Detection and Network Monitoring, ID 99* (1999), Usenix Association, pp. 29–40.

[232] SEKAR, R., GUANG, Y., VERMA, S., AND SHANBHAG, T. A high-performance network intrusion detection system. In *ACM Conference on Computer and Communications Security* (1999), pp. 8–17.

[233] SEKAR, R., GUPTA, A., FRULLO, J., SHANBHAG, T., TIWARI, A., YANG, H., AND ZHOU, S. Specification-based anomaly detection: a new approach for detecting network intrusions. In *Proceedings of the 9th ACM conference on Computer and communications security* (2002), ACM Press, pp. 265–274.

[234] SEKAR, R., AND UPPULURI, P. Synthesizing fast intrusion prevention/detection systems from high-level specifications. In *Proceedings of the 8th Usenix Security Symposium* (1999), pp. 63–78.

[235] SINGH, P. K., AND LAKHOTIA, A. Analysis and detection of computer viruses and worms: an annotated bibliography. *SIGPLAN Not. 37*, 2 (2002), 29–35.

[236] SMAHA, S. Haystack: an intrusion detection system. In *Fourth Aerospace Computer Security Applications Conference, 12–16 Dec. 1988, Orlando, FL, USA* (Washington, DC, USA, 1988), IEEE Computer Society Press, pp. 37–44.

[237] SNAPP, S., SMAHA, S., TEAL, D., AND GRANCE, T. The DIDS (distributed intrusion detection system) prototype. In *USENIX Association. Proceedings of the Summer 1992 USENIX Conference, 8-12 June 1992, San Antonio, TX, USA* (1992), Berkeley, CA, USA : USENIX Association, 1992, pp. 227–33.

[238] SNAPP, S. R., BRENTANO, J., DIAS, G. V., GOAN, T. L., HEBERLEIN, L. T., LIN HO, C., LEVITT, K. N., MUKHERJEE, B., SMAHA, S. E., GRANCE, T., TEAL, D. M., AND MANSUR, D. DIDS (Distributed Intrusion Detection System) – motivation, architecture, and an early prototype. In *Proceedings of the 14th National Computer Security Conference* (Washington, DC, 1991), pp. 167–176.

[239] SOMAYAJI, A. *Operating System Stability and Security through Process Homeostasis*. PhD thesis, University of New Mexico, 2002. `http://www.cs.unm.edu/~soma/pH/uss-2000.pdf` Accessed 31 May 2002.

[240] SOMAYAJI, A., HOFMEYR, S., AND FORREST, S. Principles of a computer immune system. In *Meeting on New Security Paradigms, 23-26 Sept. 1997, Langdale, UK* (New York, NY, USA, 1997), ACM, pp. 75–82.

[241] SOPHOS PLC. Sophos - anti-virus for business. `http://www.sophos.com/` Accessed 1 Janyary 2003.

[242] SOURCEFIRE NETWORK SECURITY. VRT certified rules for snort CURRENT, October 2005. `http://www.snort.org/pub-bin/downloads.cgi`, accessed November 10, 2005.

[243] STANIFORD, S., HOAGLAND, J. A., AND MCALERNEY, J. M. Practical automated detection of stealthy portscans. *Journal of Computer Security 10*, 1-2 (2002), 105–136. Available at `http://www.silicondefense.com/pptntext/Spice-JCS.pdf` Accessed 16 August 2002.

[244] STOLCKE, A., AND OMOHUNDRO, S. Hidden Markov Model induction by bayesian model merging. In *Advances in Neural Information Processing Systems* (1993), S. J. Hanson, J. D. Cowan, and C. L. Giles, Eds., vol. 5, Morgan Kaufmann, San Mateo, CA, pp. 11–18.

[245] STOLCKE, A., AND OMOHUNDRO, S. M. Best-first model merging for hidden Markov model induction. Tech. Rep. TR-94-003, International Computer Science Institute, 1947 Center Street, Suite 600, Berkeley, CA, 94704-1198, 1994.

[246] SYMANTEC CORPORATION. Symantec AntiVirus$^{\text{TM}}$corporate edition. `http://enterprisesecurity.symantec.com/products/products.cfm?ProductID=155` Accessed 1 Janyary 2003.

[247] TENG, H. S., CHEN, K., AND LU, S. C. Y. Adaptive real-time anomaly detection using inductively generated sequential patterns. In *IEEE Symposium on Security and Privacy* (1990), pp. 278–284.

[248] TOMBINI, E., DEBAR, H., MÉ, L., AND DUCASSÉ, M. A serial combination of anomaly and misuse IDSes applied to HTTP traffic. In *20th Annual Computer Security Applications Conference* (2004).

[249] TSUDIK, G., AND SUMMERS, R. Audes-an expert system for security auditing. *Computer Security Journal 6*, 1 (1990), 89–93.

[250] TURKIA, M. Intrusion detection systems. Available at `http://www.cs.helsinki.fi/u/asokan/distsec/documents/turkia.ps.gz` Accessed 30 July 2002., 2000.

[251] VACCARO, H. S., AND LIEPINS, G. E. Detection of anomalous computer session activity. In *1989 IEEE Symposium on Security and Privacy, 1-3 May 1989, Oakland, CA, USA* (1989), Washington, DC, USA : IEEE Comput. Soc. Press, 1989, pp. 280–289.

[252] VALDES, A., AND SKINNER, K. Adaptive, model-based monitoring for cyber attack detection. In *Recent Advances in Intrusion Detection (RAID 2000)* (Toulouse, France, October 2000), H. Debar, L. Me, and F. Wu, Eds., no. 1907 in Lecture Notes in Computer Science, Springer-Verlag, pp. 80–92.

[253] VALDES, A., AND SKINNER, K. Probabilistic alert correlation. In *Recent Advances in Intrusion Detection (RAID 2001)* (2001), no. 2212 in Lecture Notes in

Computer Science, Springer-Verlag. http://www.sdl.sri.com/papers/r/a/raid2001-pac/prob_corr.pdf Accessed 20 August 2002.

[254] VARGIYA, R., AND 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)* (Nov. 2003), pp. 50–59. Workshop held in conjunction with The Third IEEE International Conference on Data Mining. Available at http://www.cs.fit.edu/~pkc/dmsec03/dmsec03notes.pdf. Accessed 5 April 2006.

[255] VERWOERD, T., AND HUNT, R. Intrusion detection techniques and approaches. *Computer Communications 25*, 15 (September 2002), 1356–65.

[256] VIEGA, J., AND MCGRAW, G. *Building Secure Software: How to Avoid Security Problems the Right Way*. Addison Wesley, Reading, MA, 2002.

[257] VIGNA, G., ECKMANN, S., AND KEMMERER, R. Attack languages. In *ISW 2000. 34th Information Survivability Workshop, 24–26 Oct. 2000, Cambridge, MA, USA* (Piscataway, NJ, USA, 2000), IEEE, pp. 163–6.

[258] VIGNA, G., AND KEMMERER, R. NetSTAT: a network-based intrusion detection approach. In *14th Annual Computer Security Applications Conference, 7–11 Dec. 1998, Phoenix, AZ, USA* (Los Alamitos, CA, USA, 1998), IEEE Computer Society, pp. 25–34.

[259] VIGNA, G., AND KEMMERER, R. A. NetSTAT: a network-based intrusion detection system. *Journal of Computer Security 7*, 1 (1999), 37–71.

[260] VIGNA, G., ROBERTSON, W., AND BALZAROTTI, D. Testing network-based intrusion detection signatures using mutant exploits. In *Proceedings of the 11th ACM conference on Computer and communications security* (2004), ACM Press, pp. 21–30.

[261] WAGNER, D., AND DEAN, D. Intrusion detection via static analysis. In *Proceedings of the 2001 IEEE Symposium on Security and Privacy* (May 2001). Available at http://www.csl.sri.com/users/ddean/papers/oakland01.pdf. Accessed 21 April 2006.

[262] WAN, T., AND YANG, X. D. IntruDetector: a software platform for testing network intrusion detection algorithms. In *Seventeenth Annual Computer Security Applications Conference, 10–14 Dec. 2001, New Orleans, LA, USA* (Los Alamitos, CA, USA, 2001), IEEE Computer Society, pp. 3–11.

[263] WANG, K., AND STOLFO, S. J. Anomalous payload-based network intrusion detection. In *Recent Advances in Intrusion Detection: 7th International Symposium,*

RAID 2004, Sophia Antipolis, France, September 15-17, 2004. Proceedings (2004), vol. 3224 of *Lecture Notes in Computer Science*, Springer, pp. 203–222.

[264] WARRENDER, C., FORREST, S., AND PEARLMUTTER, B. A. Detecting intrusions using system calls: Alternative data models. In *IEEE Symposium on Security and Privacy* (1999), pp. 133–145.

[265] WIERS, D. Tunneling SSH over HTTP(S), 2006. `http://dag.wieers.com/howto/ssh-http-tunneling/`, Accessed 2006 Sept 13.

[266] WILLIAMS, P., ANCHOR, K., BEBO, J., GUNSCH, G., AND LAMONT, G. CDIS: Towards a computer immune system for detecting network intrusions. In *Lecture Notes in Computer Science 2212* (2001), Springer-Verlag, pp. 117–133. Presented at the 4th International Symposium on Recent Advanced in Intrusion Detection (RAID 2001). `http://en.afit.edu/ISSA/publications/WilliamsRAID.pdf` Accessed 19 August 2002.

[267] WU, S., CHANG, H., JOU, F., WANG, F., GONG, F., SARGOR, C., QU, D., AND CLEAVELAND, R. Jinao: Design and implementation of a scalable intrusion detection system for the ospf routing protocol. To appear in Journal of Computer Networks and ISDN Systems. `http://projects.anr.mcnc.org/JiNao/JiNaoJournal.ps` Accessed 30 December 2002.

[268] YE, N. A Markov chain model of temporal behavior for anomaly detection. In *Proceedings of the 2000 IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop, 2000* (2000), pp. 171–174.

[269] YE, N., CHEN, Q., EMRAN, S. M., AND NOH, K. Chi-square statistical profiling for anomaly detection. In *IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop June 6–7, 2000 at West Point, New York* (June 2000), pp. 187–193.

[270] YE, N., EHIABOR, T., AND ZHANG, Y. First-order versus high-order stochastic models for computer intrusion detection. *Quality and Reliability Engineering International 18*, 3 (May 2002), 243–50.

[271] YE, N., EMRAN, S., CHEN, Q., AND VILBERT, S. Multivariate statistical analysis of audit trails for host-based intrusion detection. *IEEE Transactions on Computers 51*, 7 (July 2002), 810–20.

[272] YE, N., AND LI, X. Application of decision tree classifiers to computer intrusion detection. In *DATA MINING 2000 Data Mining Methods and Databases for Engineering, Finance and Other Fields, July 2000, Cambridge, UK* (Southampton, UK, 2000), N. Ebecken and C. Brebbia, Eds., WIT Press, pp. 381–90.

[273] YE, N., YU, M., AND EMRAN, S. M. Probabilistic networks with undirected links for anomaly detection. In *IEEE Systems, Man, and Cybernetics Information Assurance and Security Workshop June 6–7, 2000 at West Point, New York* (June 2000), pp. 175–179.

[274] YEGNESWARAN, V., BARFORD, P., AND ULLRICH, J. Internet intrusions: global characteristics and prevalence. In *Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems* (2003), ACM Press, pp. 138–147.

[275] YEUNG, D.-Y., AND CHOW, C. Parzen-window network intrusion detectors. In *16th International Conference on Pattern Recognition, 11–15 Aug. 2002, Quebec City, Que., Canada* (Los Alamitos, CA, USA, 2002), R. Kasturi, D. Laurendeau, and C. Suen, Eds., vol. 4, IEEE Computer Society, pp. 385–8.

[276] ZANERO, S., AND SAVARESI, S. M. Unsupervised learning techniques for an intrusion detection system. In *Proceedings of the 2004 ACM symposium on Applied computing* (2004), ACM Press, pp. 412–419.

[277] ZHANG, Y., AND LEE, W. Intrusion detection in wireless ad-hoc networks. In *MobiCom 2000. Sixth Annual International Conference on Mobile Computing and Networking, 6–11 Aug. 2000, Boston, MA, USA* (New York, NY, USA, 2000), ACM, pp. 275–83.

Appendix A

Snort configuration

snort is started at system boot time by using the command:

```
snort -Dq -O -A none -c /usr/local/etc/snort.conf\
      not net 198.49.217.0/25
```

The *snort* configuration file contains the following:

```
# Configuration directives
config reference_net: 198.49.217.128/25
config logdir: /var/log/snort
config umask: 022
config show_year
config stateful

# defragment IP packets
preprocessor frag2

# TCP stream reassembly
preprocessor stream4: disable_evasion_alerts
preprocessor stream4_reassemble:\
noalerts, clientonly, ports [80]
```

```
# limit to packets inbound for web server
log tcp !198.49.217.0/25 any -> any 80\
(session: printable;)
```

The shell script that moves the collected data so it does not grow too big in a single directory is:

```
#! /bin/sh
#
# move the snort directory to avoid the "too many links" message
#
#date=`date +%d-%m-%Y`
date=`date +%Y-%m-%d`
mv /var/log/snort /var/log/snort-$date
mkdir /var/log/snort
```