ZERO-DAY WEB ATTACK DETECTION USING COLLABORATIVE AND TRANSDUCTION-BASED ANOMALY DETECTORS

by

Sharath Hiremagalore
A Dissertation
Submitted to the
Graduate Faculty
of
George Mason University
In Partial fulfillment of
The Requirements for the Degree
of
Doctor of Philosophy
Computer Science

Committee:

______________________________	Dr. Angelos Stavrou, Dissertation Director
______________________________	Dr. Daniel Barbará, Committee Member
______________________________	Dr. Duminda Wijesekera, Committee Member
______________________________	Dr. Anthony Stefanidis, Committee Member
______________________________	Dr. Sanjeev Setia, Department Chair
______________________________	Dr. Kenneth S. Ball, Dean, The Volgenau School of Engineering
Date: _________________________	Summer 2015 George Mason University Fairfax, VA

Zero-day Web Attack Detection Using Collaborative and Transduction-Based Anomaly Detectors

A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy at George Mason University

By

Sharath Hiremagalore
Master of Science
George Mason University, 2011
Bachelor of Technology (Honors)
Shanmugha Arts, Science, Technology, and Research Academy (SASTRA) University, 2007

Director: Dr. Angelos Stavrou, Professor
Department of Computer Science

Summer 2015
George Mason University
Fairfax, VA

Dedication

I dedicate this dissertation to my family for without their support, this would not have been possible.

Acknowledgments

“I can no other answer make but thanks, and thanks”

William Shakespeare

I would like to express my deepest gratitude to my committee chair, Dr. Angelos Stavrou, for his support and guidance. He persuaded me to start my doctoral program and continually engaged me to learn and conduct good research. I would also like to thank my committee members, Dr. Duminda Wijesekera, Dr. Daniel Barbará, and Dr. Anthony Stefanidis, for their valuable advice and support throughout the doctoral program. Also thanks to Dr. Dan Fleck and Dr. Walter Powell for spending countless hours working closely with me on my research projects. Their support and valuable advice has made this work possible.

Additionally, thanks to Dr. Bo Yu, Dr. Fengwei Zhang, and Dr. Konstantinos Kolias, for providing me with all the feedback and support I needed during the final months of my doctoral program. I thank Nathaniel Boggs, Quan Jia, Brian Schulte, Rahul Murmuria, and Chaitanya Yavvari for reviewing my writing and providing helpful suggestions and feedback during the course of my program. I would like to thank Patricia Carcamo who has helped me with all of the administrative work through these years. I wish to express my gratitude to all of my other colleagues at the Center for Assured Research and Engineering for their help and support.

Finally, I would like to thank my grandparents, parents, sister, aunts, uncles, cousins, and friends who have stood by me throughout this journey. They have been deeply understanding of my absence when they needed me. They have helped me keep a balanced routine by reminding me not to procrastinate, eat healthy, get enough exercise, and take breaks. Without my family’s support, this work would not have been possible.

I am truly grateful for all that my family, friends, and colleagues have done to make this dissertation possible.

Table of Contents

List of Tables

List of Figures

Abstract

ZERO-DAY WEB ATTACK DETECTION USING COLLABORATIVE AND TRANSDUCTION-BASED ANOMALY DETECTORS

Sharath Hiremagalore, PhD

George Mason University, 2015

Dissertation Director: Dr. Angelos Stavrou

Web applications have emerged as the primary means of access to vital and sensitive services such as online payment systems and databases storing personally identifiable information. Unfortunately, the need for ubiquitous and often anonymous access exposes web servers to adversaries. Indeed, network-borne zero-day attacks pose a critical and widespread threat to web servers that cannot be mitigated by the use of signature-based intrusion detection systems.

Content-based Anomaly Detection (AD) techniques are regarded as a promising mechanism to detect 'zero-day' attacks. AD sensors have also been shown to perform better than signature-based systems in detecting polymorphic attacks. However, the False Positive Rates (FPRs) produced by current AD sensors have been a cause of concern.

In the first part of this work, we present a collaborative approach to quickly detect zero-day attacks. To detect previously unseen attacks, we correlate web requests containing user-submitted content across multiple web servers that is deemed abnormal by local Content Anomaly Detection (CAD) sensors. We are the first to propose the exchange of suspicious (abnormal) request content between sites, which significantly reduces false positives.

The cross-site information exchange happens in real-time leveraging privacy preserving data structures. Moreover, we automatically filter out high entropy and rarely seen legitimate requests, reducing the amount of data and time an operator has to spend sifting through alerts. Our results come from a fully working prototype using eleven weeks of real-world data from production web servers. During that period, we identify at least three application-specific attacks not belonging to an existing class of web attacks as well as a wide range of traditional classes of attacks, including SQL injection, directory traversal, and code inclusion without using human-specified knowledge or input.

In the second part of this work, we introduce and evaluate transAD, a system of payload inspection AD sensors that are based on Transductive Confidence Machines (TCM). Existing TCM-based implementations have very high false positive rates and are not suitable for use as NIDS.

Our approach leverages an unsupervised machine learning algorithm to identify anomalous packets; unlike most AD sensors, ours does not require manually labeled data. Also, transAD uses an ensemble of TCM sensors to achieve better detection rates and lower FPRs than single sensor implementations. Therefore, transAD presents a hardened defense against poisoning attacks.

We evaluated our prototype implementation using two real-world data sets collected from a public university's network. Approximately 1.1 million packets containing real attacks were processed by our AD sensor. To compute the ground truth, we manually labeled 18,500 alerts. In the course of scanning millions of packets, our sensor's low false positive rate would significantly reduce the number of false alerts that need to be inspected by an operator while maintaining a high detection rate.

Chapter 1: Introduction

1.1 Motivation

Web-based solutions are one of the most popular and convenient technologies to provide any type of service to consumers, businesses, and governments alike. Web-based products are ubiquitous and have been driving the e-commerce industry since the tech boom in the late 1990s [1]. As web services have become increasingly popular and play an important role in carrying personally identifiable information, securing these services has become a paramount priority for the industry. Signature-based Network Intrusion Detection Systems (NIDSs) is one solution that has been typically used to protect web servers from attacks. However, these systems are unable to detect new attacks that have never been seen before, known as “zero-day” attacks.

Anomaly Detection (AD) systems work by learning the baseline network traffic being monitored. Any deviation from this baseline network traffic is flagged as an alert by an AD system. This gives AD systems an edge over other NIDSs that rely on known signatures for detecting attacks. Therefore, AD systems present a promising solution to identify zero-day attacks and are also able to detect polymorphic attacks to web-based services. Polymorphic attacks are variants of existing attacks for which signatures are not yet available.

Although AD systems have existed for years and have the advantage of detecting zero-day and polymorphic attacks, they have not been widely adopted by the industry for production use as NIDSs. This is because current AD systems have high False Positive Rates (FPRs). As each alert needs to be manually processed by an IDS operator, high FPR makes them less practical. Another drawback of current implementations of AD systems is that they require supervised training data. This requires labor-intensive labeling of the training dataset. Finally, Anomaly Detection-based Intrusion Detection Systems are susceptible to poisoning attacks. These attacks can deceive the learning algorithm as they slowly inject malicious traffic that resembles baseline traffic.

1.2 Contributions

In this dissertation, we try to alleviate the shortcomings of the current Anomaly Detection-based Intrusion Detection Systems. We present a collaborative web-based intrusion detection system approach to quickly identify zero-day and polymorphic attacks and reduce the number of false positives an operator manually inspects. We also present novel Anomaly Detection-based IDS for the web. The main contributions made by this work are summarized below:

- We propose and deploy a collaborative Anomaly Detection system that exchanges and correlates alerts generated by individual AD systems at each site.
- We show and analyze empirical evidence supporting the benefits of collaborative Anomaly Detection-based NIDS. The collaborating systems are exchange content alerts generated at each site. From our experiments, we are able to achieve zero-day attack detection with a very small false positive rate.
- Next, we expand the collaborative Anomaly Detection-based NIDS to three sites and demonstrate the ability to detect zero-day and polymorphic attacks while further decreasing the false positive rate at every collaborating site.
- We developed a novel Anomaly Detection-based Network Intrusion Detection system for the web that is based on unsupervised learning and does not require any labeled training data.
- We applied a novel combination of our hash-based distance metric, unsupervised AD system, and ensemble-based techniques to produce outstandingly low false positive rates.

- Furthermore, our novel AD system is able to detect never-before-seen zero-day attacks. Our experiments use real-world datasets to show that the system successfully detects the following class of attacks: Buffer Overflow, Code Injection, File Inclusion, Directory Traversal, Script and Polymorphic Attacks, and others.

1.3 Organization

In the first part of this dissertation, we present a collaborative approach of Anomaly-based Intrusion Detection system to quickly detect widespread zero-day attacks. In Chapter 2 we describe the experimental results of exchanging content alerts across two domains. We present our results and analysis of the common alerts produced at the two sites. In Chapter 3 we extend the previous study of collaborative intrusion detection to three sites and present results over a larger dataset. In the second part of this dissertation (Chapter 4), we present our novel Anomaly Detection-based intrusion detection technique for the web and discuss our conclusions and future work.

Chapter 2: Experimental Results of Cross-Site Exchange of Web Content Anomaly Detector Alerts[1]

2.1 Introduction

Automated zero-day and polymorphic attacks pose a critical widespread threat to web servers. Network-based Anomaly Detection (AD) is regarded as a potential defense to this very difficult-to-identify threat. However, AD sensors often suffer excessive false positive rates that require an unacceptable amount of human effort to properly resolve. When an alert is generated, the operator does not have a well-defined "attack signature" to analyze —she must drill down to packet content in order to understand the nature and validity of the alert. Therefore, to distinguish a true attack from a legitimate but anomalous packet, operators have to manually sift through the alerts and offending packet content. On the other hand, any attempt to reduce the rate of false positives by modifying the sensitivity of the AD sensor may reduce the true positive detection rate.

In this chapter, we present results using a working alert exchange architecture with an extensive section on model comparisons over an extended period of time. Currently, to limit the false positives, AD sensor outputs are typically correlated with other evidence to distinguish true attacks from false alarms. For instance, see shadow servers in [2]. *We propose a large-scale network of AD sensors distributed across disjoint domains that exchange and correlate web server content alerts to identify widespread zero-day attacks in real time.* This network would analyze ingress traffic to some sample of collaborating web servers. Sensors could be deployed at each domain or at a common peering point. The zero-day attacks found could then be communicated broadly.

[1]This work was done in collaboration with Nathaniel Boggs and Salvatore Stolfo, Department of Computer Science, Columbia University, N.Y.

The results from an initial deployment of the system across two administrative domains on the Internet support the feasibility and accuracy of such a system. Additionally, we present a method of comparing normal models between sites to potentially identify sites with distinct normal traffic flows. We conjecture that each administrative domain may detect zero-day attack vectors as abnormal content since, by definition, zero-day attacks are data delivered to a service that have not been seen before and are not contained in a signature database. Each site will also classify some legitimate traffic as abnormal thus generating false positives. However, it is likely that this traffic will only be seen at a single site: errors of this nature are not likely to be similar at different domains because normal traffic flows will be different. The same (or nearly the same) abnormal packet content seen at two or more sites is most likely a widespread attack vector rather than a false positive. Hence, correlating abnormal data across two or more sites in real-time may detect and accurately identify zero-day attacks. In the presented results, we manually confirm the attack vectors in our correlated alerts. The number of zero-day attacks specifically depends on which signature engine the attacks are compared to. Furthermore, we claim that real-time filtering of zero-day attacks against web servers is feasible with essentially no human intervention by automatically filtering common abnormal content.

The strategy to correlate common abnormal content will detect zero-day attacks that do not use sophisticated polymorphic engines. In the case of polymorphic attacks, where each infection produces an entirely new version of the attack for each propagation attempt [3], it is unlikely that this cross-domain correlation strategy will work. One should not expect to see any common attack vectors. In those cases, correlating AD alerts with host-based instrumented shadow servers is likely a better strategy to detect zero-days and reduce false positives. However, as it stands today, most of the web-based attacks we have detected deliver their payload as relatively short PHP arguments and do not contain polymorphic attack engines nor have we seen code attempting to download polymorphic variants. Hence, we posit that the cross-domain correlation strategy is effective against the large class of zero-day attacks targeting web-based applications and services.

To validate our claims, we study the outcome of two weeks of real network data capture and an automated exchange of AD alerts between Columbia University and George Mason University over the Internet. Our empirical results confirm our theory: the more distinct each normal model may be, the more likely it is for common AD alerts to identify and filter true zero-day attacks. Indeed, by comparing the normal models from different domains we establish that each site has a distinct model of normal content. Moreover, throughout the two-week study, we found 11,787 common alerts. Furthermore, we analyze the time between each site first detecting each attack and verify that real-time exchange is feasible. With this baseline of common content-based web server Anomaly Detector alerts composed almost entirely of attacks, the few false positives can be quickly identified and shared to reduce the human workload. These experimental results support our conjecture that cross-domain content-based AD correlation deployed at a large scale could effectively detect and mitigate zero-day web attacks.

2.2 Related Work

Previous work on distributed intrusion detection has focused mainly on the exchange of data within a single organization. Much of the early work, *e.g.,* [4, 5] focused on limited distribution within an enclave. In [6], the authors discuss methods for cooperatively correlating alerts from different types of intrusion detection systems. Krugel *et al.* [7, 8] concluded that only a relatively small number of messages (seldom more than two) need to be exchanged to determine an attack is in progress, making decentralized intrusion detection feasible and appropriate. DShield [9] is the most active volunteer-based DIDS project on the Internet that we are aware of, focusing on "top 10"-style reports and blacklists; however, it uses a centralized model, relies on reports from volunteers, and generally scrubs data. In [10, 11] the authors describe more general mechanisms for node "cooperation" during attacks.

DOMINO [12] is probably the closest decentralized framework in this scope to ours. The paper measured, using DShield alert logs, the notion of information gain; however, DOMINO does not incorporate alerts generated by Anomaly Detectors. In addition, Farroukh *et al.* proposed a distributed and collaborative intrusion detection system called DaCID [13] based on the Dempster Shafer theory of evidence of fusing data. Additionally, in [14] the authors used a decentralized analyzer. Tian *et al.* introduced an alert correlation model based on hierarchical architecture [15].

The AD system we employ is based on STAND by Cretu *et al.*, [2] a derivative of an earlier system called Worminator [16]. A collaborative technique where the sites exchange abnormal models to improve detection was presented in [2]. In [17], the authors completely automated the process of determining the optimal AD sensor parameters for a single sensor. As user interactions with systems change over time, the current model becomes stale and may incorrectly classify new traffic patterns [18]. However, in all of the above systems, the authors did not explore the benefits and caveats of exchanging anomaly detector content alerts. This paper provides the novel contribution that cross-domain AD alert exchange can identify zero-day attack vectors.

2.3 Experiment Architecture

We call our complete system AutoSense (Figure 2.1), which consists of an expanded Worminator [16] alert exchange and storage system integrated with deployed local AD sensors to exchange alerts and abnormal models in real time. Using a client-server architecture, each administrative domain has a Worminator client install that receives the raw alerts and models from a sensor. Each client then encodes alerts by inserting the n-grams of the content into Bloom filters [19] as needed before sending the alerts and models to the server over an encrypted channel. On the Worminator server, alerts and models are stored in a database. Separate threads perform correlation on the stored alerts, continuously matching all non-private content from the host domain to all Bloom filter representations of private alerts.

We use the combination of STAND [17], an automated training and sanitizing process,

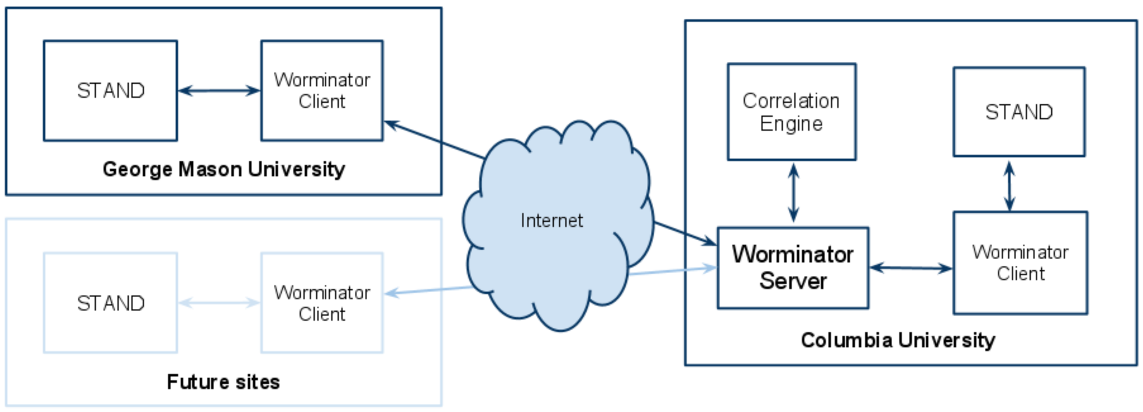


Figure 2.1: Architecture

with the Anagram sensor [20] as our network-based Anomaly Detection sensors. We have sensors deployed at Columbia University and George Mason University inspecting inbound network traffic. To allow a proper training period and still have time for the data exchange to reveal common alerts, we collected traffic over the period of two weeks. The sensors' automated training process requires around 50-60 hours. For testing, we ran our sensors on TCP port 80 traffic inbound to two web servers: *www.cs.columbia.edu* and *www.gmu.edu.* For the collection, aggregation, and correlation of the HTTP data, we used VMWare virtual machines running Ubuntu Server 9 64bit. Each virtual machine was equipped with 16GB ram and 2-4 CPUs. The AD sensors are designed to sample a subset of packet traffic by parsing and normalizing TCP port 80 GET request URIs. In order to reduce variability, we strip the URI down to just the string of arguments and then remove numbers and decode hex characters. The resulting normalized argument strings that are fewer than 17 characters are dropped as attack vectors and are much longer. This subset of packet content allows our machines to still see a wide range of potential attacks as numerous applications have a web service front-end. For each alert we log the IP address, timestamp, and content string.

After each site ran the sensor on its two weeks, of data from the same time period, we correlated the resulting alert content strings.

2.4 Model Comparison

We theorize that correlating AD content alerts between sites with distinct normal traffic flows will reduce the false positive rate since legitimate requests will be less likely to be similar. For a large-scale system, we will want to minimize the alert comparisons between similar sites to prevent more false positives. Therefore, a method to quickly compare the similarity of normal traffic between sites will be vitally important. In this section, we use each site's normal model as an approximation of its normal traffic flow for cross-site comparisons.

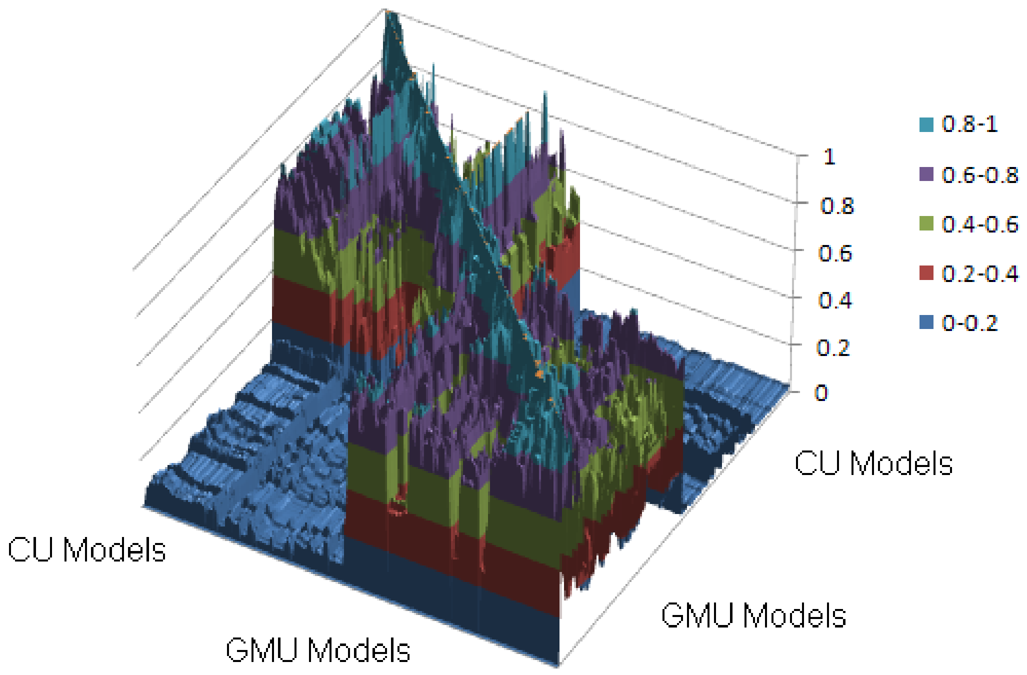


Figure 2.2: Normal Models Compared across Site and Time

In order to explain our model comparison results, here is a brief explanation of the model generation process. For a complete description, please see previous work on STAND [2]. STAND has a sanitizing process where small micro-models are continuously created on small sets of data once little new traffic is seen, generally around 3 hours depending on data volatility. After 25 of these micro-models are created, they use a voting process to test all the data the micro-models used. All the data is then voted on by micro-models. Data that passes the vote is added to a Bloom filter to create a now sanitized, normal Anagram model. Once the first sanitized model is created, then the process is repeated each time a new micro-model is made. The new micro-model replaces the oldest one so that the latest 25 micro-models are always used to create a new sanitized Anagram model.

Each Anagram model is a Bloom filter with a 2^{28}-long bit array initialized to 0s. Data is added to the model by hashing n-grams from the data into the bit array, setting some bits to 1. While not an exact measurement, we believe, since the same hash functions are used for all models, that directly comparing the bits that are set in each Bloom filter gives a general idea of how similar two models are. In all our comparisons we find the number of 1 bits that both Bloom filters share and divide by the total number of 1 bits in the Bloom filter. This comparison operation C is represented below:

Let B1, B2 be Bloom filters from Anagram models with bits $\{1...i...2^{28}\}$.

C(B1,B2) = (Number of bits i such that $B1[i] = 1$ and $B2[i] = 1$) / (Number of bits i such that $B1[i] = 1$)

Each AD sensor computes models of normal data for consecutive epochs at each site, producing a set of time-ordered models. Fig. 2.2 displays each individual normal model compared against other normal models, both of which are from the same and collaborating site. The top quadrant shows Columbia University models compared to one another, while the bottom quadrant shows George Mason University models compared to one another. The models are placed in time order from top to bottom. Notice that the content flows at each site slowly change over time. The comparison of the time-ordered models computed at the same site shared 49% of set bits. Furthermore, the left side of the graphic in Fig. 2.2

shows George Mason University models compared to Columbia University models, and the right side shows Columbia University models compared to George Mason University models. Notice that the content flows at each site are truly distinct. The important lesson is that the CU and GMU anomaly detectors have learned entirely different "normal" models at each site.

With an average of 5% rate of common bits set between models from separate sites, we show that both sites are quite diverse. This confirms the intuition that distinct sites have radically different models of normal data and supports the ability of Anomaly Detectors to recognize attack traffic from distinct domains. With distinct models, traffic detected as abnormal at separate sites is much more likely to be an attack. Attackers attempting to put together a mimicry attack against multiple sites likely face an impossible task. They would have to target it specifically to one site since the sites have diverse normal content flows.

2.5 Alert Correlation

The initial correlation process began with 41,232 alerts observed at Columbia University and 20,678 at George Mason University. We compare each local alert to each Bloom filter encoded alert from the remote site until we find a match. To account for simple polymorphism that could exist in the alerts, we consider a match to be 80% of n-grams from the local alert being present in the Bloom filter and the alert lengths to be within 80% of one another. We found 11,787 common alerts, 7,989 at Columbia University and 3,798 at George Mason University.

We confirm our online results from the Bloom filter comparisons with an offline study using the Levenshtein string distance [21]. We normalized this distance by the length of the longer string to find equivalent alerts. The Levenshtein algorithm is a simple and effective way to correlate content alerts offline. For more information on suitable algorithms for content correlation see [22]. After testing, we set the threshold for matching at 0.2. This means that we allow for up to two changes in every 10 characters. In addition, the unique

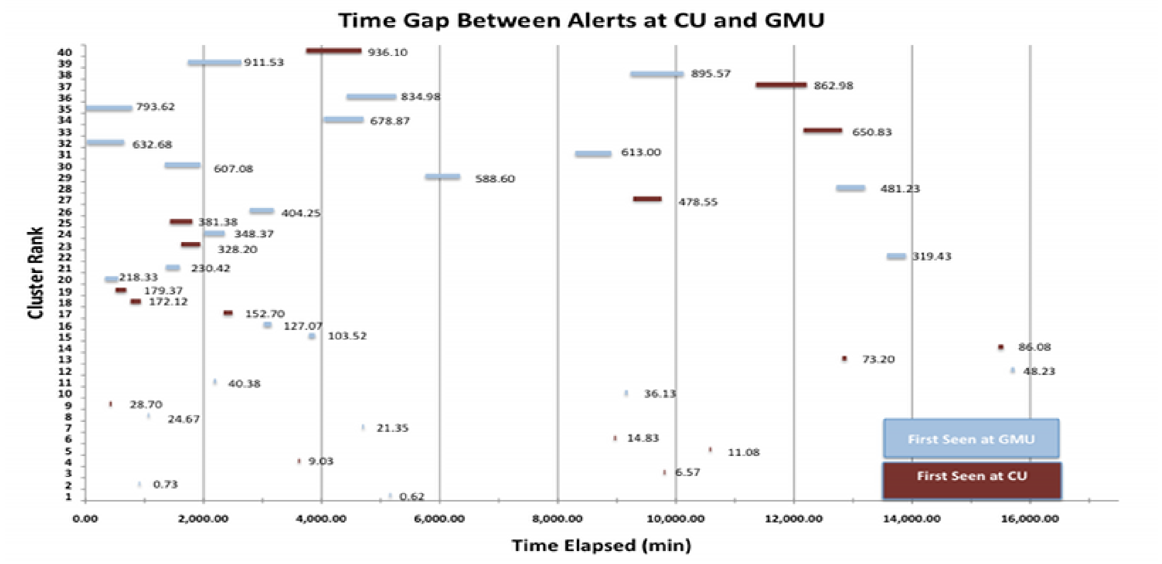


Figure 2.3: Time gap between Common Alerts at CU and GMU for 40 shortest time gap clusters.

sets of alert content strings from each site were clustered using the normalized Levenshtein distance to obtain the common alerts. This correlation results in 96 common content alert string clusters representing 12,353 total alerts, 8,570 at Columbia University and 3,783 at George Mason University. Using these two different methods and seeing similar results confirms that encoding the alerts in Bloom filters still allows for accurate correlation.

Using manual inspection, we see that all but four of the attack clusters are indeed true attacks in both correlation methods. An Internet Explorer automated browser request related to an office toolbar made up 2,554 alerts. This alert is caused by an IE browser extension and will likely be seen as an anomalous request by all web servers. This distinct but highly repetitive alert should be identified as a false positive once upon its first occurrence and subsequently ignored by simply filtering it as "noise." Since it is so common, the false positive rate drops precipitously after filtering this specific alert. Three additional clusters of false positives produces a net false positive rate of 0.69% out of common alerts and $1.2 * 10^{-5}\%$ (68 of 549 millions packets) out of total incoming packets to the web servers. For this dataset, covering a two-week period, a human operator would have had to manually inspect 96 clusters to identify the three false positives. This is strong evidence that cross-domain alert exchange is a valuable security measure with minimal amounts of human interaction needed. Out of the 3 false positives, two are iframe tags and the other is related to a Twitter feed. This makes intuitive sense as one of the only ways that legitimate traffic would appear at both sites would be from a generic request with never-before-seen parameters. Nevertheless, manual inspection of a false positive cluster from one operator could save all other sites from having to identify the alert. We believe that other false positives will most likely also be fairly generic mistakes. Therefore, the limited number of these generic argument strings will be identified as false positives, and then any future results will have an even lower false positive rate.

Now that we have the alerts common to both sites clustered, we analyze the timing of alerts. An interesting measurement is the time lag between each cluster's first detection at each site. Fig. 2.3 shows the time lag, which varied widely from the shortest delay being

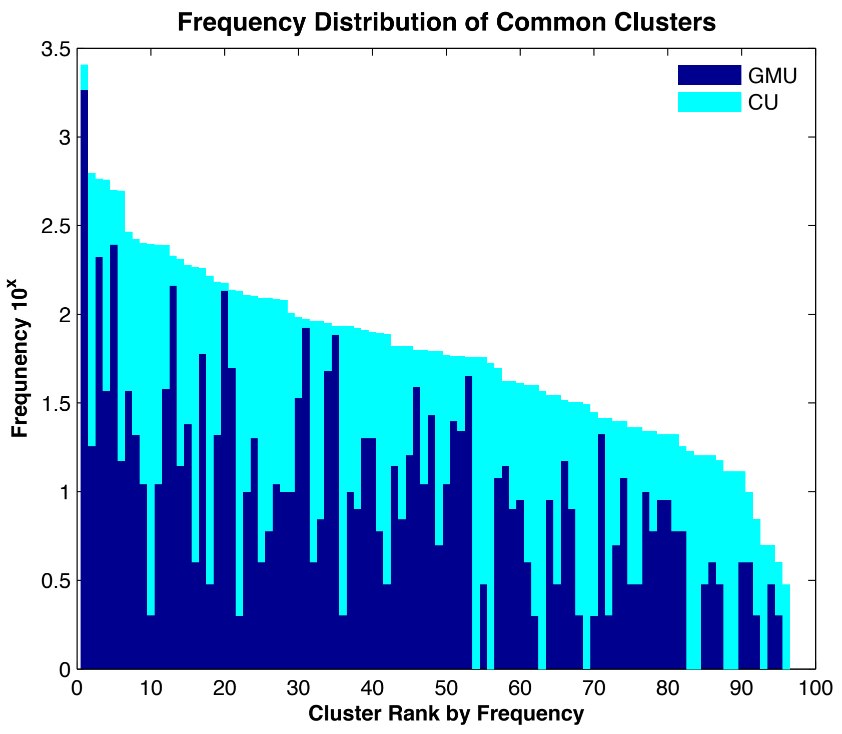


Figure 2.4: Frequency of Common Alerts —Normalized Levenshtein distance

37 seconds to attacks being seen at the second site after 8 days. Without information from a third site, we cannot directly estimate the utility of broadcasting "filters" to many other collaborating sites. Nor can we measure the impact of filters on reducing the infection rate of a large-scale attack. Hence, with additional collaborating sites we may directly measure the response time needed to provide wide-area protection against an attack. Extrapolating the data from the two sites, we hypothesize that a real-time exchange of a watch list with confirmed attack signatures would be able to filter even more rapid attacks. Given that the large number of attacks fit into a small number of clusters, generating a signature for each cluster seems feasible. We also measured the duration of each alert cluster. Our results show that some attacks persist beyond the time that both sites have seen them. This suggests

that a watch list still benefits the sites that first identify a new attack.

Furthermore, we compute the frequency distribution of the common alert clusters for Columbia University and George Mason University. In Fig. 2.4 we can see that the presented distribution of alerts follows an exponentially decreasing trend, indicating that a small fraction of the alert clusters are responsible for the majority of the total alerts. This favors our collaborative approach because we can mark only a small number of dense clusters at one of the sites to sift through the majority of the alerts. The rest of the small frequency clusters can be vetted over time since their rate of appearance is also small, thus manageable. Currently, our collaborative architecture is applied on feeds at two sites. This makes it difficult to estimate how the alert clustering and frequency distribution would change with the addition of other collaborating sites.

AutoSense can be employed as a means of extracting zero-day attacks from web application streams at a peering point or any set of distributed sensors across enterprises. The entire set of packet streams need not be analyzed, but rather an AD model may be computed from a sample of ingress packets destined to a selected web server. By comparing the models, a group of "collaborating" servers may then be chosen, from which a pool of correlated common anomalous web requests would then be extracted by AutoSense. Those are likely zero-day attacks as evidenced by the Columbia University and George Mason University experiments. With the addition of more peers, the process of exchanging and marking alert clusters is going to require a more comprehensive approach for operator synchronization and prioritization. This warrants further investigation in our future work.

But where do these alerts come from? Are they also clustered in terms of network location? In order to answer the above questions, we tried to identify the origin of the common alerts. To that end, we computed the frequency of common alerts by country of origin. Initially, we mapped each IP address to the country of registration using the IP geolocation tool [23]. Then, we computed the frequency distribution of IP addresses causing alerts. The map in Fig. 2.5 shows the frequency distribution of alerts at Columbia University and George Mason University from IP addresses common to both sites. The United States

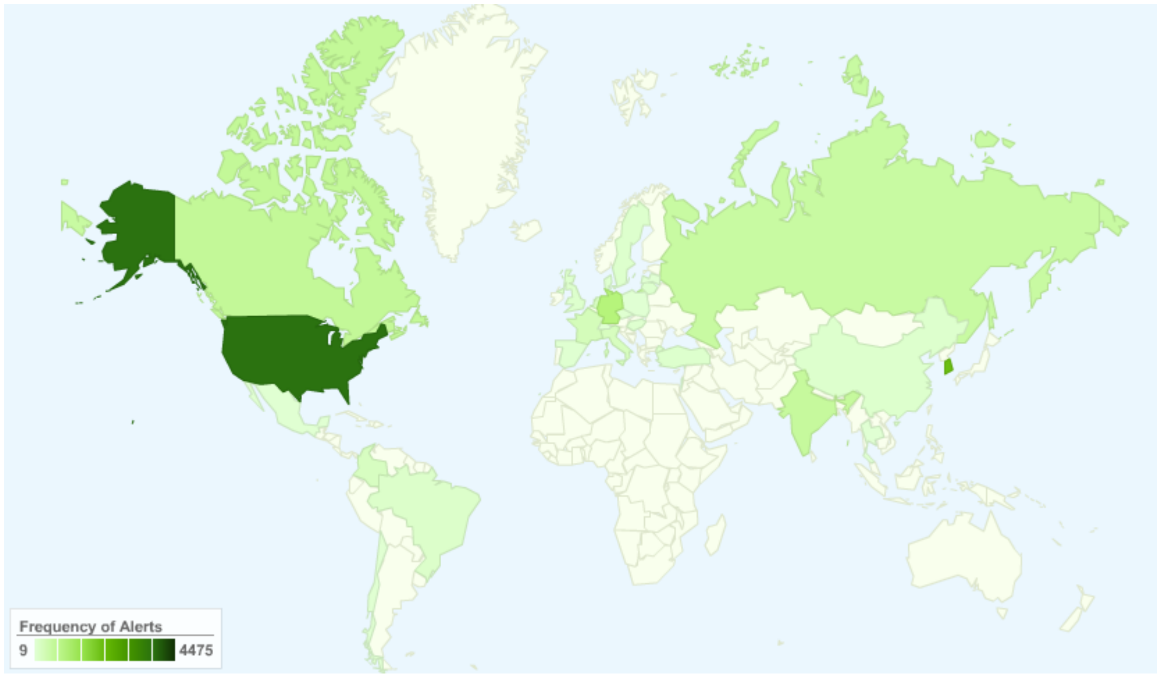


Figure 2.5: Frequency of Common Alerts by Country

account for the majority of the common alerts, totaling 4,475 attacks followed by South Korea at 2,581 alerts. The alerts common to both Columbia University and George Mason University originate from 27 different countries. Thus, the sources of the common alerts are concentrated in a limited number of high-density countries.

2.6 Conclusions

We present and analyze empirical evidence supporting the benefits from deploying a distributed content-based Anomaly Detection system. Our findings demonstrate the potential for efficient large-scale mitigation of the zero-day attacks and false positives by real-time filtering of common attacks. Indeed, a total of 11,787 alerts were confirmed by both AD systems over a period of two weeks. Our correlation of real alerts between distinct sites demonstrated that, in most cases, we can boost the detection performance by identifying attack clusters and false positives in one of the sites ahead of time. With this number of alerts between only two sites, we posit that, if our system is expanded to a large-scale, a significant portion of zero-day web attacks could be identified and mitigated. Our findings support our theory that collaborative cross-domain content-based AD correlation might be a potential solution to the web-based zero-day attacks.

Chapter 3: Cross-domain Collaborative Anomaly Detection: So Far Yet So Close[1]

3.1 Introduction

Web applications are the primary means of access to the majority of popular Internet services including commerce, search, and information retrieval. Indeed, online web portals have become a crucial part of our everyday activities with usage ranging from bank transactions and access to web-based email to social networking, entertainment, and news. However, this reliance on ubiquitous and, in most cases, anonymous access has turned web services into prime targets for attacks of different levels of sophistication. Newly crafted attacks, often termed "zero-day," pose a hard-to-address challenge compromising thousands of web servers before signature-based defenses are able to recognize them [24]. Although recent research indicates that Anomaly Detection (AD) sensors can detect a class of zero-day attacks, currently, AD systems experience limitations that prevent them from becoming a practical intrusion detection tool.

In this paper, we propose a new defense framework where Content Anomaly Detection (CAD) sensors, rather than traditional IDS systems, share content alerts with the aim of detecting widespread, zero-day attacks. Contrary to pure alert correlation and fusion [25], we exchange abnormal content across sites as a means to reduce the inherent high false positive rate of local CAD systems. To that end, we show how local CAD sensors can be leveraged to generate an accurate, reliable alert stream where false positives are consumed through a process of alert validation; false positives rarely make their way to a human operator. Further, we implement information exchange mechanisms, enabling the collaborative

[1]This work was done in collaboration with Nathaniel Boggs and Salvatore Stolfo, Department of Computer Science, Columbia University, N.Y.

detection of attacks across administrative domains. We believe such collaboration, if done in a controlled and privacy-preserving manner, will significantly elevate costs for attackers at a low cost for defenders. Our system has a number of core capabilities: high-quality, verified alert streams that focus on detecting the presence of and learning from zero-day attacks and previously unseen attack instances; scalable alert processing; and modular multi-stage correlation. Figure 3.1 illustrates the overall architecture.

Intuitively, inbound web requests fall into three categories: legitimate low-entropy requests, legitimate high-entropy or rarely seen requests, and malicious requests. Legitimate low-entropy requests are the most accurately modeled by CAD systems. Therefore, each individual CAD sensor will label previously seen, low-entropy requests as normal and will not exchange them with other CAD sensors. Legitimate high-entropy or rare requests will often show up as abnormal to the local CAD sensor and will therefore be exchanged. Since remote sites do not have similar content due to the high entropy nature or rarity of these requests, no matches will be identified, and thus no alerts will be raised. On the other hand, malicious requests will appear as abnormal in many local CAD models. Therefore, when exchanged, they will match other sites and alerts will be raised. The more sites participating, the better the coverage and faster the response to widespread web attacks. Space and structural constraints due to HTTP protocol and specific web application parsing limit the ability for an attacker to fully exploit polymorphism techniques, analyzed in [26], so each zero-day attack should exhibit similar content across the attacked web services.

In our experimental evaluation, we use 11 weeks of traffic captured from real-world production web servers located in different physical and network locations. We do not inject any artificial or additional data. All attacks and statistics described are observed on live networks. We measure the detection and false positive changes from adding an additional server in the sharing system. Most interestingly, we confirm the theory presented by [27] that false positives tend to repeat across sites. Additionally, as most of the false positives occur early and often, we show that CAD systems can benefit greatly from a reasonable cross-site training period. This reduces the number of false positives to 0.03% of all the

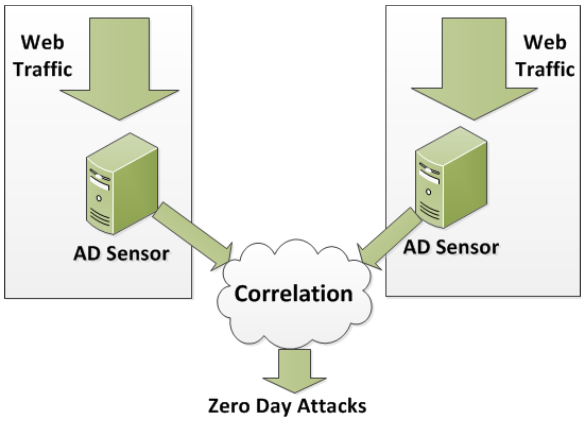


Figure 3.1: Architecture

normalized web requests. Furthermore, we quantify the similarity of the produced CAD models from each site over long periods of time. Using these models, we provide an analysis of how aggregate normal and abnormal data flows compare between sites and change over time. Additionally, we furnish results regarding the threshold of the matching content and the effects of increasing the set of participating collaborating sites. Finally, we are the first to present a real-world study of the average number of alerts a human operator has to process per day. Moreover, we show that the alert sharing and correlation of alerts reduces the human workload by at least an order of magnitude.

3.2 Related Work

Anomaly Detection techniques have been employed in the past with promising results. Alexsander Lazarevic *et al.* compares several AD systems in Network Intrusion Detection [28]. For our analysis, we use the STAND [2] method and Anagram [20] CAD sensor as our base CAD system. The STAND process shows improved results for CAD sensors by introducing a sanitization phase to scrub training data. Automated sensor parameter tuning has been shown to work well with STAND in [17]. Furthermore, the authors in [18] observe that replacing outdated CAD models with newer models helps improve the performance of the sensor as the newer models accurately represent the changes in network usage over time. In all of the above works, due to limited resources within the domain, a global picture of the network attack is never examined. Additionally, to achieve manageable false positives rates, a single site AD sensors could be used at best as a filter to reduce the load to an additional computationally expensive shadow server that performs dynamic execution of suspicious data as proposed in [20].

Intrusion Detection Systems that leverage both matching or even machine-learning techniques suffer from well-known limitations [29]. In the past, there has been a lot of criticism for Anomaly Detection techniques [30] especially focusing on the high volume of the false positives they generate. With our work, we dispel some of this criticism and we show that we can improve the performance of CAD systems by sharing content information across sites and correlating the content alerts.

Initially, Distributed Intrusion Detection Systems (DIDS) dealt with data aggregated across several systems and analyzed them at a central location within a single organization. EMERALD [4] and GrIDS [5] are examples of these early scalable DIDS. Recent DIDS systems dealt with collaborative intrusion detection systems across organizations. Krügel *et al.* developed a scalable peer-to-peer DIDS, Quicksand [7, 8], and showed that no more messages than twice the number of events are generated to detect an attack in progress. DShield [9] is a collaborative alert log correlation system. Volunteers provide their logs to DShield where they are centrally correlated, and an early-warning system provides "top 10"-style reports and blacklists to the public free of charge. With the rise of botnets, IP address black lists become less and less useful as attacks can originate from multiple sources. Our work differs in that we rely on the actual user-submitted content of the web request rather than the source IP. More general mechanisms for node "cooperation" during attacks are described in [10, 11].

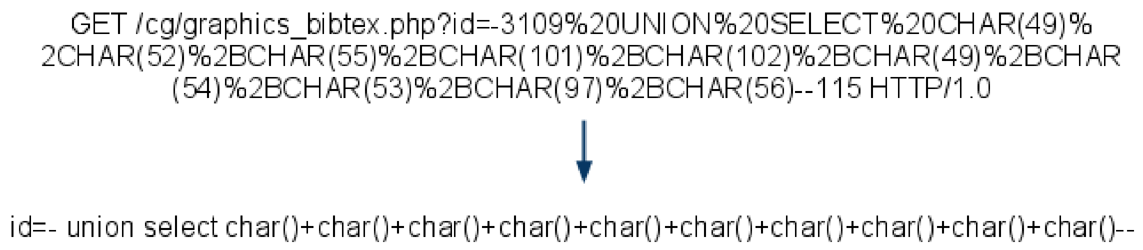
DOMINO [12], a closely related DIDS, is an overlay network that distributes alert information based on hash of the source IP address. DShield logs are used to measure the information gain. DOMINO differs from our technique as it does not use AD to generate alerts. DaCID [13] is another collaborative intrusion detection system based on the Dempster Shafer theory of evidence of fusing data. Another DIDS with a decentralized analyzer is described by authors in [14].

Centralized and decentralized alert correlation techniques have been studied in the past. The authors in [15] introduce a hierarchical alert correlation architecture. In addition to scalability in a DIDS, privacy preservation of data send across organizations is a concern. Privacy preservation techniques that do not affect the correlation results have been studied. A privacy preserving alert correlation technique, also based on the hierarchical architecture [31], scrubs the alert strings based on entropy. We expand Worminator [16], a privacy-preserving alert exchange mechanism based on Bloom filters, which had previously been used for IP alerts. Furthermore, Carrie Gates *et al.* [32] used a distributed sensor system to detect network scans albeit showing limited success. Finally, there has been extensive work in signature-based intrusion detection schemes [33, 34]. These systems make use of packet payload identification techniques that are based on string and regular expression matching for NIDS [35–37]. This type of matching is only useful against attacks for which some pattern is already known. Our system is capable of detecting a much wider range of zero-day attacks as we do not rely on prior knowledge.

3.3 System Evaluation

3.3.1 Data Sets

We collected contiguous eight weeks of traffic between October and November 2010 of incoming HTTP requests to two popular university web servers: *www.cs.columbia.edu* and *www.gmu.edu*. In addition, to measure the effects of scaling to multiple sites, we added a third collaborating server, *www.cs.gmu.edu*. This results in a second dataset with an additional three weeks in December 2010 of data from all three servers. The second data set allows us to analyze the effects of an additional web site to the overall detection rate and network load. To that end, we are able to show the change in the amount of alert parsing a human operator would have to deal with in a real-world setting and analyze models of web server request content. All attacks detected are actual attacks coming from the Internet to our web servers and are confirmed independently using either IDS signatures [34, 38] developed weeks after the actual attacks occurred and manual inspection when such signatures were not available. However, that does not preclude false negatives that could have been missed by both signature-based IDS and our approach. The number of processed packets across all of our datasets are over 180 million incoming HTTP packets. Only 4 million of them are deemed potentially suspicious because our normalization process drops simple web requests with no user submitted variables.



```
GET /cg/graphics_bibtex.php?id=-3109%20UNION%20SELECT%20CHAR(49)%
2CHAR(52)%2BCHAR(55)%2BCHAR(101)%2BCHAR(102)%2BCHAR(49)%2BCHAR
(54)%2BCHAR(53)%2BCHAR(97)%2BCHAR(56)--115 HTTP/1.0
```

```
id=- union select char()+char()+char()+char()+char()+char()+char()+char()+char()+char()--
```

Figure 3.2: A normalization example from a confirmed attack. The first line of the original GET request is shown. We use the output of the normalization function for all future operations.

3.3.2 Normalized Content

Our system inspects normalized content rather than other common attributes such as frequency or source IP address. In practice, repeated attacks from the same IP address can easily expose the attacker. This is why sophisticated attackers that want to remain stealthy employ botnets that offer multiple IP addresses to perform large-scale attacks. We focus our attention to this class of sophisticated adversaries and their attacks web applications using specially crafted user content.

To that end, we process all HTTP GET requests and extract all user-defined content (*i.e. user specified parameters*) from the URI across all request packets. Putting aside serious HTTP protocol or server flaws, the user-specified argument string appears to be the primary source of web attacks. We use these user-specified argument strings to derive requests that are deemed abnormal and can be used for correlating data across servers serving different pages. Additionally, we normalize these strings in order to more accurately compare them [27, 39]. In addition, we decode any hex-encoded characters to identify potential encoding and polymorphic attacks. Any numeric characters are inspected but not retained in the normality model to prevent overtraining from legitimate but high-entropy requests. Also, we convert all the letters to lowercase to allow for accurate comparisons and drop content fewer than five characters long to avoid modeling issues. Figure 3.2 illustrates this process.

Beyond GET requests, we perform tests analyzing POST request data as well. POST requests are approximately 0.34% of the total requests. However, our experiments show that the current CAD sensor does not accurately train with data that exhibits large entropy typical in most POST requests. We leave the development of a new CAD sensor that can accurately model POST requests for future work and focus on analyzing GET requests, which dominate the web traffic we observe (99.7%).

3.3.3 Content Anomaly Detector and Models

In cross-site content correlation, each site builds a local model of its incoming requests using a Content Anomaly Detection (CAD) sensor. In our experiments, we leverage the STAND [2] optimizations of the Anagram [20] CAD sensor although any CAD sensor with a high detection rate could be used with our approach. However, we apply the CAD sensors on normalized input instead of full packet content as they originally operated on in order to obtain more accurate results. Additionally, we fully utilize all of the automatic calibration described in [17], including the abnormal model exchange to exclude repeated attacks from poisoning the training data. The Anagram normal models are, as described in [20], Bloom filters [19] containing the n-gram representation of packets voted as normal by the STAND micro-models. A Bloom filter is a one-way data structure where an item is added by taking multiple hashes and setting those indices of a bit array to one. This provides space efficiency and incredible speed suitable for high-speed networks since adding an element or checking if one is already present are constant time operations. Each normalized content is split into 5-gram sections as in [2] using a sliding window of five characters. See Figure 3.3 for an example. Requests can then be easily tested as to how many of the n-grams from their argument string are present in the model. N-grams give us a granular view of content allowing partial matches as opposed to hashing full content while maintaining enough structure of the content to be much more accurate than character frequency models. Previous work [16] calibrated Bloom filters to have an almost non-existent false positive rate and shows that extracting the content is infeasible, which allows for the preservation of privacy. The models we use are 2^{28} bits long and compress to about 10-80KB, a size that is easily exchanged as needed. The Anagram models test whether new content is similar to previous content by comparing how many of the n-grams exist in the model already. New unseen content is then always considered abnormal.

3.3.4 Alert Exchange

We use a modular design for our content alert and abnormal model exchange system. We leverage the initial work of Worminator [16], an alert exchange system that we heavily extend to meet the needs of our system. A content exchange client instance runs at each site and receives content alerts and abnormal models. We use a common format so that any CAD sensor can easily be adapted to work with the exchange system. In our case, each site's local STAND/Anagram sensor sends the content alert packet to the Worminator client as soon as it tests a packet and finds it abnormal. The Worminator client then encodes the

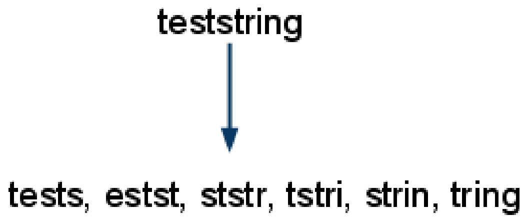


Figure 3.3: Each string is broken up into n-grams to be hashed into a Bloom filter as part of a model of traffic or in order to preserve privacy. The n-gram representation allows for partial matches instead of having to rely on exact string comparisons.

content alerts as Bloom filters if at a remote site and then sends the content alerts and any abnormal models through a secure channel over the Internet to the Worminator server. The bandwidth usage with this alert exchange turns out to be minimal since we only look at GET requests with argument strings and then further narrow down content by only exchanging the abnormal content alerts. It turns out that each alert encoded in a Bloom filter takes about 2KB to transmit on average. For our eight-week experiment, this translates into an average of 0.9Kb/sec bandwidth needed per site for a real-time system, leaving plenty of room to scale up to a large set of collaborators before running into bandwidth constraints. A back-end process on the server performs the correlation of content alerts by comparing the local unencoded alerts to the Bloom filter representation of alerts from remote sites. We perform all our experiments faster than real-time while exchanging encoded content alerts securely over the Internet.

By exchanging content alerts from remote sites only in their Bloom filter form, our system can protect the privacy of legitimate web requests. During the Bloom filter correlation process, only the fact that a match occurs can be determined —not any specific content. If a match occurs then this is a piece of content that a site has already seen coming to their server, so the only new information revealed is that the other site also had this content

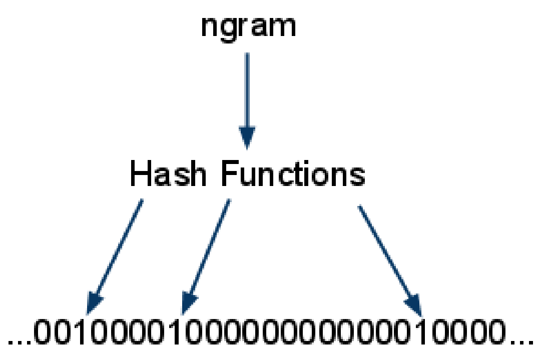


Figure 3.4: Each n-gram of the input string is hashed into the Bloom filter setting a number of bits to true. Future n-grams can be hashed, and then the bits are checked to see if all are true. If so, then the n-gram is assumed to have been previously added.

incoming. In this way we can gain information about the content we have in common, which most likely represents attacks, while keeping the remaining content private in case there is sensitive information in the web requests.

3.3.5 Scaling to Multiple Sites

Our initial system deployment consists of three web servers. However, to be even more effective at quickly detecting widespread attacks, we envision a larger-scale system deployment consisting of many collaborating sensors monitoring servers positioned in different locations on the Internet. Of course, for the system to scale up to include more sites, the correlation and alert comparison process must scale accordingly. Indeed, if we consider the pair-wise comparison of local alerts with each remote alert, it appears to grow asymptotically: $O(n^2)$. This can quickly become a problem; however, we can bound this total computation under a constant K by varying the amount of time duplicate alerts that are stored in the system.

In practice, we did not observe problems during our experiments, even keeping eight weeks of data for correlation, because indexing of alerts can be done using existing computationally efficient algorithms. Moreover, we only have to operate on unique alerts which are much smaller in size. Additionally, if a longer time frame is desirable, we can employ compression to the remote site alerts into a small number of Bloom filters by trading off some accuracy and turning the scaling into order O(n), allowing many more alerts to be stored before running into any scaling issues. In that case, each time a new site joins the collaboration, our local site must compare its alerts to the Bloom filters of those from the new site. Therefore, the overall computational complexity scales linearly with the number of remote sites participating. Since we can bound the local comparison with a remote site under K, the total computational cost scales linearly as well, and each site has optional tradeoffs in time alerts kept and Bloom filter aggregation if local resources are limited. In practice, based on our numbers even with an unoptimized prototype, we could scale to around 100 similar servers operating in real-time and comparing all alerts over a few weeks' time. If additional utility is derived from having even more participating servers, then optimizing the code, time alerts kept, and trading off accuracy in Bloom filter aggregation should easily allow additional magnitudes of scaling.

3.4 Model Comparison

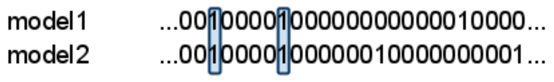

model1 ...0010000100000000000010000...
model2 ...0010000100000010000000001...

Figure 3.5: Each model is stored in a Bloom filter. We count the number of set bits in common and then divide by the total number of set bits.

Each normal model is a Bloom filter with all the n-grams of each normal normalized request added to it. By comparing Bloom filters as bit-arrays, we are able to estimate how much content models share. We test how many set bits each pair of models have in common and divide by the total number of set bits to get a percentage of set bits in common. The generated Bloom filters are quite sparse; therefore, the overlap of bits between content should be small as observed in Figure 3.5. We use this model comparison metric to measure server distinctness and change in normal flows over time; whether servers in the same domain share additional commonality; and how much abnormal data we see in common across servers.

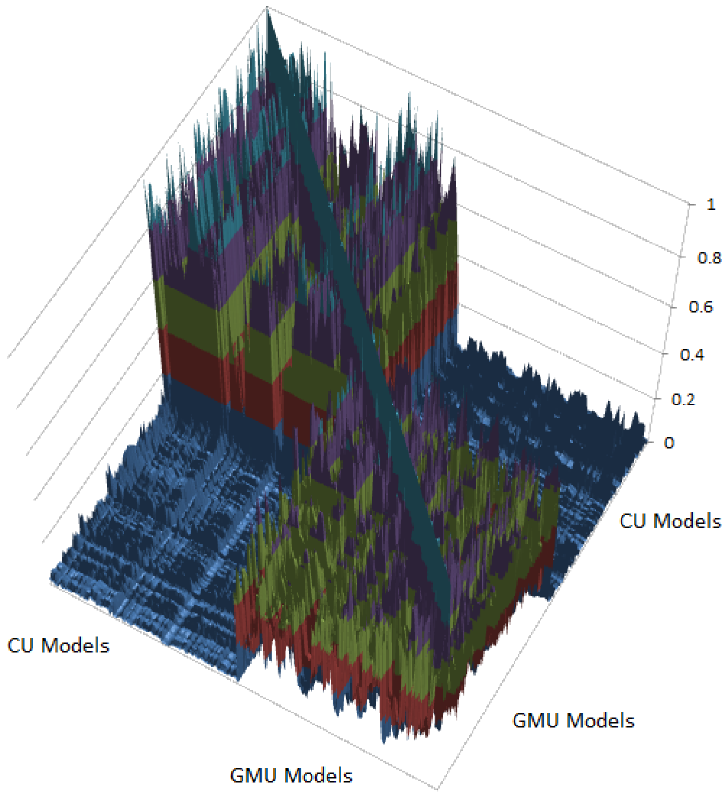


Figure 3.6: Model comparison: The higher the figure, the higher percentage of set bits the models had in common. The top and bottom quadrant are intra-site comparisons. The sides represent the comparison across sites which, as expected, appear to have differences.

We first use this comparison to observe the differences in models from distinct sites with one another. We took every fifth model from our runs and compared the ones from the same runs to their counterparts at the other location. For normal models in our eight-week run, we see on average 3.00% of bits set in common. Compare this to the over 40% of bits in common on average comparing models at the same site. See Table 3.1. This shows that there is some overlap, indicating that not filtering out normal content before performing the content alert correlation could lead to increased false positives, but that for the most part, each site maintains a distinct core of content over time. While we do not have enough sites to measure how important this distinctness is to the accuracy achieved via correlation of alerts, we do confirm that at least for distinct web servers, our correlation process achieves effective results. See Figure 3.6 for a visual of the model comparison results. We find that models across long periods seem to keep a core of commonality but differ more than models close together in time. A product of this gradual change appears even with only five weeks difference in our datasets'; averaged over eight weeks, both sites keep more than 40% of bits in common while in the three-week run this is closer to 50%. This reinforces existing work [2] showing that traffic patterns do evolve over time, indicating that updating normal models periodically should increase effectiveness.

Comparison	**Normal Models Oct.-Nov.**	**Abnormal Models Oct.-Nov.**
Columbia CS	41.45%	52.69%
GMU Main	41.82%	38.51%
Cross site	3.00%	10.14%

Table 3.1: Commonality of normal and abnormal models.

With our three-week dataset, we also have an additional web server from one administrative domain. With two web servers from the same Autonomous System, we compare them to each other to see if our work has the potential to help a large organization that may have many separate web servers, even if collaboration with other organizations proves too difficult. See Table 3.2 and Table 3.3 for empirical details. Interestingly, we find no more commonality among normal models in the same domain than across domains. The fact that abnormal models at these web servers share about as much in common with the server from another domain as one another suggests that attackers likely do not specifically target similar IP ranges with the same attacks. This suggests that web server administration and location may not play a factor in the effectiveness of using a particular web server for collaboration. An organization with sufficiently distinct web servers might be able to achieve good results without having to overcome the obstacles related to exchanging data between organizations.

Comparison - Normal Models	**Columbia CS**	**GMU Main**	**GMU CS**
Columbia CS	44.89%	3.89%	4.08%
GMU Main		48.39%	2.41%
GMU CS			56.80%

Table 3.2: Comparison of normal models between three sites. (Percentages of set bits in common shown.)

The abnormal models from different sites show some similarity with close to 10% set bits matching, while models from the same site show more similarity. The high amount of common abnormal data between models at the same site may be influenced by legitimate requests classified as abnormal. More interesting is the commonality across sites. These shared bits most likely represent the attack data that we have found in common. There is an irregularity where some of the abnormal models are empty and could not be compared. We remove these empty models before computing the averages to avoid divide-by-zero issues. Multiple runs with the data confirm the strange behavior, so our best guess is that a strange convergence of the STAND micro-models simply voted to include all the data from that time period into the normal models, leaving the abnormal models empty.

Comparison - Abnormal Models	**Columbia CS**	**GMU Main**	**GMU CS**
Columbia CS	53.05%	9.46%	9.32%
GMU Main		48.39%	8.55%
GMU CS			70.77%

Table 3.3: Comparison of abnormal models between three sites. (Percentages of set bits in common shown.)

Overall, our model comparisons provide quite interesting results. We find that each site has normal traffic flows that are distinct although changing somewhat over long periods of time. We see no major distinctions in comparison of same domain servers versus servers on separate domains, which indicates that our system could be deployed by a large organization to protect itself without having to rely on outside collaborators. Finally, our measurements of abnormal data validate the idea that separate servers will receive similar attacks.

3.5 Correlation Results

The correlation process compares each unique content alert from the local sensors against the Bloom filter representation of each unique content alert from other sites. If at least 80% of the n-grams match the Bloom filter and the length of content before encoding, which is also exchanged, is within 80% of the raw content, then we note it as a match. The number

cx=:cjygsheid&cof=forid:&ie=utf-&q=machine+learning+seminar&sa=go
o=-&id=&s=uhkf&k=hsbihtpzbrxvgi&c=kg

Table 3.4: Normalized examples of legitimate abnormal data seen at only one site.

can be optimized further depending on how many false positives human operators are able to manage. These matches are what the system identifies as attacks. Once these attacks are identified, the Bloom filter representation could be sent out to any additional participating servers and future occurrences could be blocked. In order to confirm our correlation results with the Bloom filters, we also perform an offline correlation of results using string edit distance [21] with a threshold of two changes per 10 characters. We cluster together any pair of alerts from either site with less than this threshold. If a cluster contains alerts from more than one site, then it represents a common content alert. With only minor differences, these methods give us similar performances, confirming that using privacy-preserving Bloom filters provides an accurate and computationally efficient correlation. To simulate a production deployment, we use the Bloom filter comparison as our default correlation technique and use the string edit distance clustering only to facilitate manual inspection as needed, especially at a single site. See Table 3.4 for examples of true negatives where legitimate requests are not alerted on since each is seen at just one site. Table 3.5 shows an example of the same attack with a slight variation being matched between two sites.

We run our experiments correlating the abnormal traffic between sites from our October-November eight-week dataset and our December three-week dataset and then manually classify the results since ground truth is not known with an in-the-wild dataset such as ours. We display our classification of our system's alerts in Table 3.6. As we predicted in [27], most of the false positives repeat themselves early and often, so we also display the

faq=' and char()+user+char()= and "='
id=' and char()+user+char()= and "='

Table 3.5: Normalized example of abnormal data; one from each site that matched.

	Oct-Nov	**Oct-Nov with training**	**Dec.**	**Dec. with training**	**Gained by adding third server**	**Dec. Common to Three Sites**
Duration of testing[2]	54 days	47 days	19 days	12 days		
Total false positives	46,653	362	40,364	1,031	1,006	0
Unique false positives	64	13	48	5	3	0
Total true positives	19,478	7,599	7,404	2,805	186	322
Unique true positives	351	263	186	89	9	8

Table 3.6: Main experiment results considering a match to be at least 80% of n-grams being in a Bloom filter. Note that the 5th column results are included in column 4. Also, note that due to self-training time for the CAD sensor, actual time spent testing data is about two days less.

results assuming a naïve one-week training period that labels everything seen in that week and then ignores it. While this training technique could certainly be improved upon, we choose to show this naïve example in order to better show the effectiveness of the approach as a whole and to preclude any optimizations that might turn out to be dataset specific. Such a training period provides a key service in that most false positives are either due to a client adding additional parameters regardless of web server, such as with certain browser add-ons, or servers both hosting the same application with low enough traffic throughput

[2]Due to equipment outages, approximately three hours of data is missing from the Oct.-Nov. www.cs.columbia.edu dataset and less than 0.5% of the Dec. dataset abnormal data totals are missing.

that it fails to be included in a normal model. Many of these cases tend to be rare enough to not be modeled but repeat often enough that a training period will identify them and prevent an operator from having to deal with large volumes of false positives. Certainly with such a naïve automated approach, attacks will not be detected during this training period, but after this period we end up with a large benefit in terms of few false positives with little negative beyond the single week of vulnerability. Any attacks seen during training that are then ignored in the future would have already compromised the system, so we do not give an attacker an advantage going forward. In fact, this training period serves an operator well in that many of the high-volume attacks that are leftover "background radiation" will be seen in this training period and thus not needed to be categorized in the future. Adding an additional web server in our last experiment provides a glimpse at how broadening the scope of collaboration to a larger network of web servers can help us realize a high detection rate.

Let us now analyze how accurate our system is. The false positive rate is relatively easy to measure. We manually classify the unique alerts and then count the total occurrences of each. With regard to the number of requests that pass through the normalization process, the false positive rate is 0.03%. If you calculate it based on the total incoming requests then it is much less. The true positive rate or detection rate is much harder to accurately measure since we have no ground truth working with a real dataset. First, remember what class of attacks we are trying to detect. We are trying to detect widespread attacks and leave the goal of detecting attacks targeted at a single site to other security methods in order to better leverage collaboration. With this in mind, there exists two places where a widespread attack could be missed. An attack could arrive at multiple sites but not be detected as abnormal by one of the local CAD sensors and therefore never be exchanged with other sites. The other possibility is that an attack could be abnormal at both sites but different enough that the correlation method fails to properly match it.

In the first case where a local CAD sensor fails to identify the attack as abnormal, we have a better chance to measure our accuracy. Most CAD sensors are vulnerable to mimicry attacks where an attacker makes the attack seem like normal data by padding the malicious data in such a way as to fool the sensor. We can mitigate this by deploying very different sensors to each site, which as a whole are very difficult to bypass, while individually they are vulnerable to a specific padding method. In this way, an attacker might bypass some sites, but as the attack is widespread, eventually two of the CAD sensors that the attacker is not prepared for can detect the attack and broadcast a signature out to the rest of the sites.

	Oct-Nov	**Oct-Nov with training**	**Dec.**	**Dec. with training**	**Gained by adding third server**	**Dec. Common to Three Sites**
Total false positives	47,605	439	41,845	1,017	4	0
Unique false positives	77	23	55	5	1	0
Total true positives	25,042	10,168	9,594	3,272	254	293
Unique true positives	488	362	221	109	10	8

Table 3.7: Experiment results considering a match to be at least 60% of n-grams to be in a Bloom filter.

In the latter scenario, we have to rely heavily on the vulnerable web applications having some structure to what input they accept so that attacks exploiting the same vulnerability will be forced to appear similar. We can certainly loosen correlation thresholds as seen in Table 3.7 as well as come up with more correlation methods in the future. In practice, this is where the lack of ground truth hinders a comprehensive review of our performance. As far as we can tell, between the structure imposed by having to exploit a vulnerability with HTTP parameters, lower correlation thresholds, and finding additional attributes for correlation, we should have a good head start on attackers in this race. At the very least, our layer of security will make it a race instead of just forfeiting to attackers immediately once a vulnerability is found. Without ground truth, we cannot be sure that we detect all widespread attacks. We have seen no indication in our data that attackers are using any of the above evasion techniques yet, so we believe that our system will provide an effective barrier, one which we can continue to strengthen using the above approaches.

mosconfig_absolute_path=http://phamsight.com/docs/images/head??
config[ppa_root_path]=http://phamsight.com/docs/images/head??
option=com_gcalendar&controller=../../../../../../../../../../../../../../../proc/self/environ%
id=' and user=–
id=-.+union+select+–
command=createfolder&type=image&currentfolder=/fck.asp&newfoldername=test&uuid=
option=com_user&view=reset&layout=confirm

Table 3.8: Normalized examples of actual attacks seen at multiple sites.

We detect a broad range of widespread attacks, with some examples shown in 3.8. Common classes of attacks show up such as code inclusion, directory traversal, and SQL injection. Our system faithfully detects any wide spread variants of these attacks, some of which might evade certain signature systems; however, the novel attack detection our system provides lies with the last two examples shown. These two attacks are attempting to exploit application-specific vulnerabilities, one attacking an in-browser text editor and the other a forum system. Since attacks such as these resemble the format of legitimate requests and lack any distinct attribute that must be present to be effective, existing defenses cannot defend against zero-day attacks of this class. The fact that our system caught these in the wild bodes well for its performance when encountering new widespread zero-day attacks.

An examination of the false positives explains the repeated nature and sporadic occurrences of new false positives. See Table 3.9 for some examples of normalized false positives. All the false positives fall into one of two broad categories: rare browser specific requests or rarely used web applications installed on two or more collaborating servers. For example, the most common false positive we see is an Internet Explorer browser plug-in for Microsoft

ul=&act=&build=&strmver=&capreq=
c=&load=hoverintentcommonjquery-color&ver=ddabcfcccfadf
jax=dashboard_secondary
feed=comments-rss

Table 3.9: Normalized examples of false positives seen at multiple sites.

Office which sends a GET request to the web server regardless of user intent. The use of this plug-in is rare enough that the request shows up as abnormal at all sites. As for server-side applications, we see most of the unique false positives relating to the administrative functions of isolated WordPress blogs, which see so little use that the requests stand out as abnormal. New false positives will continue to occur in small numbers as web servers and browsers evolve over time (less than one per three days on average during our eight-week run). We believe that identifying these few rare occurrences is quite manageable for operators. This task gets easier since as the number of collaborators grow, so do the resources for the minimal manual inspection needed to identify these isolated occurrences.

Adding a third web server, www.cs.gmu.edu, to the collaboration shows that additional web servers help us to identify more attacks and allows some basic insight into what types of web servers might be best grouped together for collaboration. Assuming our training method, adding this third server as a collaborating server exchanging data with www.cs.columbia.edu allows us to detect 11.25% more unique attacks than just correlating alerts between www.cs.columbia.edu and www.gmu.edu. This increase over the 80 unique attacks we detect without it supports the need for adding substantial numbers of collaborators to increase the detection rate. Unfortunately, this new collaborating server also introduces false positives that we do not see in previous experiments. We expect as with previous false positives that future experiments will most likely repeat these with few new additions. An offline correlation using edit distance shows both GMU web servers having a number of attacks in common as well. This supports an idea that collaborating with distinct web servers could be as useful as collaborating across sites. False positives seem to be a function of rarely used web services located at each server, so servers hosting only a few clearly defined and well-used services may give substantially better results.

This additional web server also provides the opportunity to require alerts to be seen by at least three sites before reporting them as attacks. While this proposition is hard to accurately evaluate with only one dataset and just three servers, of which www.cs.gmu.edu experiences much lower traffic volumes, a couple of interesting results stand out. As expected, both false positives and true positives drop off significantly. We see no false positives after the training period. This shows that for at least our datasets, all of the server-side services that cause false positives drop out once we require three web servers to have the data in common. If this continues to be the case as more servers are added, then only reporting attacks that target three or more servers could solve most of the false positive issues. While requiring three servers to confirm an attack does yield fewer true positives, the ones it does detect are quite widespread and if the collaboration is expanded, the detection should increase greatly. This method, while scaling in detection rate more slowly than only requiring two servers to confirm attacks, could be a much more effective option to keep false positives low once enough servers collaborate.

Time Gap in Minutes	**Across Site CU, GMU and GMU CS**	**Across Site CU and GMU**	**Across Site CU and GMU CS**	**Across Site GMU and GMU CS**
Min	0.23	1.48	7.52	0.23
Max	17501.07	25911.00	20589.02	24262.13
Average	4579.85	5477.35	7048.07	6489.08
Std. Dev.	5250.04	6173.61	7038.27	7634.13

Table 3.10: Time gap statistics across three sites

We measure the implications of changing the threshold for matching two alerts. Increasing the threshold past 80% to require perfect or almost perfect matches fails to help in reducing the false positives, since at this threshold almost all of the false positives are exact matches, so even requiring all n-grams to match a Bloom filter exactly does not help. Reducing the threshold to allow more loose matches does show a trade-off in increased detection of attacks at the expense of additional false positives. By only requiring 60% of n-grams from one alert to match the Bloom filter representation of another site's alert, we can expect to capture attacks with significantly more variance, such as similar payloads targeting different web applications. See experiment details in Table 3.7. While at first, the results from a lower threshold appear quite good in terms of raw numbers of alerts, looking at only the new unique alerts that human operators have to classify tells a more balanced story. Going from an 80% threshold to 60% for our eight-week run with a training period increases the detection of new unique attacks by 37.6%, while increasing the newly seen unique false positives by 76.9%. In the three-week run, the lower threshold adds no new unique false positives, pointing to the need for threshold optimization once the system scales up. In fact, it lowers the utility of adding a new server since the existing ones detect additional attacks without it. However, as the number of web servers collaborating increases, this matching threshold along with the number of servers required to share an alert before reporting it as an attack should be key settings in order to optimize the system.

From the offline generated alert clusters, we conduct a temporal study of the alerts seen across the three servers. Firstly, we look at the time gap between alerts across sites. We compute the pair-wise time gap of common alert clusters across the three servers. Additionally, we calculate the minimum time gap between alert clusters common to all three servers. Table 3.10 summarizes the minimum, maximum, average, and standard deviations of the time gaps for the above cases. A better visual representation of the common alert clusters across all three servers is represented in Figure 3.7. The graph shows the minimum time gap between alerts observed at one server and the same alert being observed at the other two servers. The horizontal axis denotes the relative time elapsed since the start of

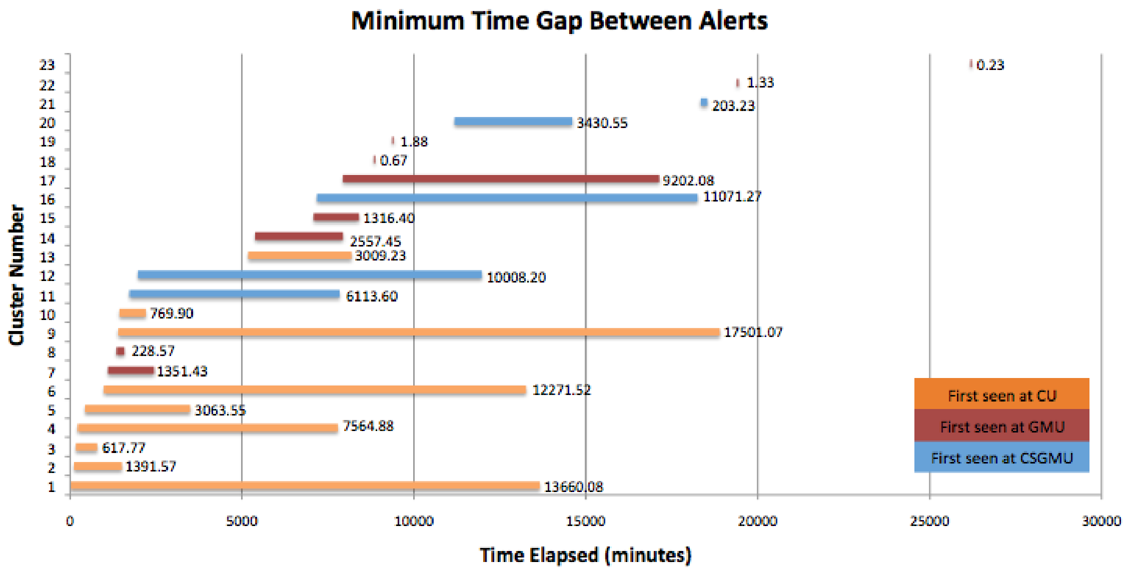


Figure 3.7: Time gap between alerts at collaborating sites.

observing the first alert. The vertical axis denotes the cluster. Each of the bars in the graph start at the time when an alert is observed at a site and ends at a time when it is seen first among the other two sites. The bar graphs are color-coded to represent where the attack was first seen. From the statistics it can be seen that the average time gap between alerts could be used to our advantage. The results from the time gap analysis from the October-November run computed across CU and GMU shows a similar large average value (Min: 1.57min, Max: 71022.07min, Average: 15172.53min, Std. Dev.: 18504.44min). This gives us sufficient time to take preventive action at the collaborating sites by exchanging a small blacklist.

Furthermore, we analyze the number of unclassified unique alerts that an operator has to manually classify every day. Figure 3.8 depicts the number of unique alerts generated daily. The graph shows both true positive and false positives observed using our collaborative

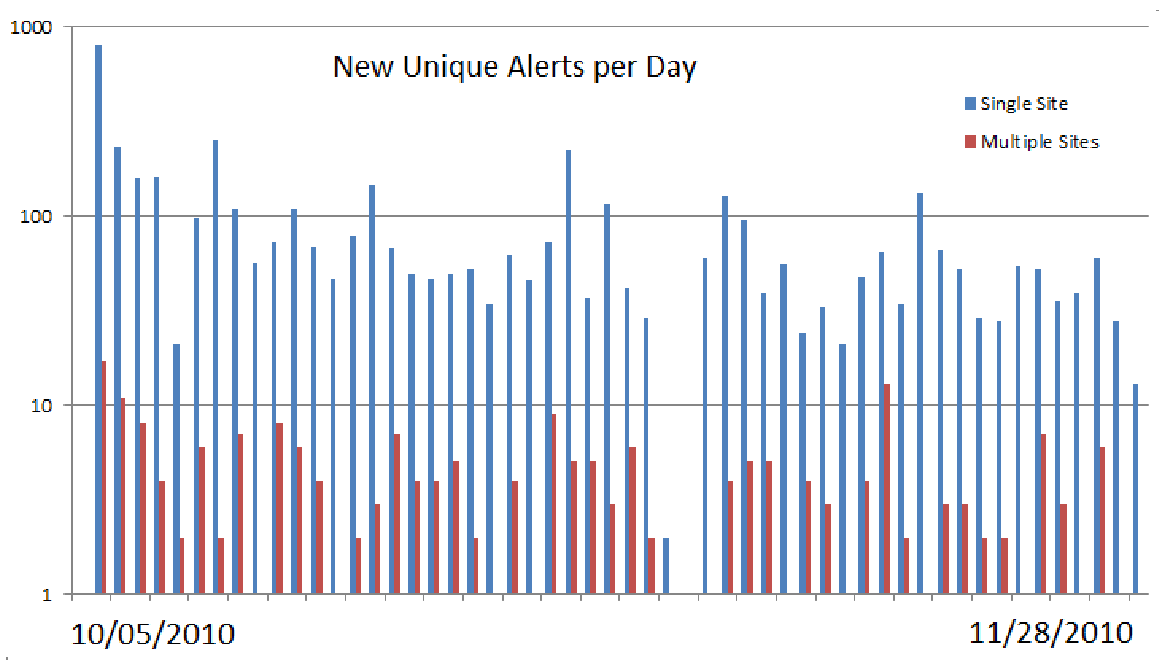


Figure 3.8: Number of new unlabeled unique alerts per day that a human operator would have to parse. The large number of false positives from the AD system is reduced by almost a magnitude difference when correlated to other sensors.

approach alongside a standalone approach. The horizontal axis denotes time in one day bins, and the vertical axis denotes the frequency of alerts observed on a log scale. For the standalone CAD sensor, a unique alert is included in the frequency when it is first observed at a site. However, for multiple sites collaborating, an alert is included in the frequency count at the time when it is confirmed to be seen at all sites. On average, the number of unique alerts observed every day using a standalone CAD sensor at CU is 82.84 compared to 3.87 alerts when using a collaborative approach, over an order of magnitude in difference. Therefore, a collaborative approach clearly reduces the load on the operator monitoring alerts to an easily managed amount.

3.6 Conclusions

Web services and applications provide vital functionality but are often susceptible to remote zero-day attacks. Current defenses require manually crafted signatures which take time to deploy, leaving the system open to attacks in the meantime.

Our work demonstrates that we can identify zero-day attacks by correlating Content Anomaly Detection (CAD) alerts from multiple sites while decreasing false positives at every collaborating site. Indeed, with a false positive rate of 0.03% the system could be entirely automated or operators could manually inspect the less than four new alerts per day on average that we observe in our eight-week experiment. We demonstrate that collaborative detection of attacks across administrative domains, if done in a controlled and privacy-preserving manner, can significantly elevate resources available to the defenders exposing previously unseen attacks.

Chapter 4: transAD: An Anomaly Detection Network Intrusion Sensor for the Web

4.1 Introduction

Web based applications have become an integral part of our lives, providing us a platform to accomplish various essential tasks. For example, Internet banking, e-commerce, and e-prescription services exchange personally identifiable information that need to be safeguarded. Additionally, many widely deployed out-of-the box web product solutions have become easy and profitable targets for attackers. Recently, attacks on these web-based applications have been on the rise [40, 41]. Although existing Network Intrusion Detection Systems (NIDS) play an important role in preventing attacks on web services, the rising number and new types of attacks highlight the need for improved web defenses.

Industry relies primarily on signature-based NIDS [38, 42–46] to secure their networks. These systems rely on previously known signatures to identify attacks. Without prior known signatures, signature-based NIDS are unable to detect new attacks, i.e., "zero-day" attacks. TransAD is one of a class of NIDS that use algorithms based on Anomaly Detection (AD) to combat zero-day attacks. AD sensors analyze traffic and identify deviations from "normal" patterns as potential attacks. However, designing these promising systems presents several challenges. First, implementations generally rely on highly labor-intensive labeled training data as a basis for establishing normal traffic patterns [47]. Second, high False Positive Rates (FPRs) are seen as a major drawback due to the amount of effort required to analyze false positives [48]. Third, AD sensors can potentially be subject to poisoning attacks that intentionally modify the learned normal model to allow various types of attacks [49]. Current AD sensors have failed to completely address all of these issues, and therefore, have not been widely deployed in the industry [47, 48].

In this chapter we present transAD —a new, self-learning, ensemble-based AD system for the web that solves all three problems. Unlike other AD systems, our system features a completely unsupervised learning algorithm that does not require any labeled training data. Given the volume of data processed by a NIDS, labeling training data for NIDS requires a prohibitive amount of manual effort and is a time-consuming process. Additionally, most of the supervised learning algorithms learn a static model of traffic and are unable to adapt over time to changes in the system. Adapting to changing patterns of traffic requires additional sets of manually labeled training data at regular intervals for re-training the supervised AD sensor. This additional time and effort required to keep supervised AD sensors updated makes their use impractical in the real world. Consequently, these methods have been heavily criticized [47]. Our method does not use labeled training data and works with network traffic collected without any manual intervention. Therefore, our method does not require any manual effort to label a training dataset; transAD is capable of continuously self-adapting to the changes in the system without additional manual intervention or effort.

In addition to obviating the manual effort and time required for labeling training datasets, our system is designed to generate low false positive rates. TransAD does this by using a combination of ensemble of sensors called micro-models, an unsupervised AD algorithm, and our hash-based distance metric. Each micro-model uses the unsupervised AD algorithm to identify potential attacks. To do so, each packet is evaluated by all of the micro-models independently. Decisions made by individual micro-models in the ensemble are combined using a voting mechanism. The use of an ensemble gives robustness to the determination of attacks (those that are considered anomalous by a majority of micro-models) and serves to reduce the errors, false positive rate in particular.

It is important to note that the direct application of the comparable AD algorithm for NIDS produces unacceptably high FPRs of approximately 3% [50]. Our AD sensor, on the other hand, yields an average false positive rate of 0.28% when evaluated using two real-world data sets. It is the judicious combination of the AD technique, with ensemble methods and a proper distance metric, that allows us to obtain demonstratively outstanding low false positive rates and high detection rates in the evaluation section of this paper. The novelty of our approach comes from the successful combination of the above techniques.

Additionally, our inherent system design makes it very difficult for perpetrators to defeat our system using poisoning attacks. In order to poison our system, the attacker needs to fabricate packets that look like normal traffic while containing elements of the intended attack, fooling the system into including these packets in the learned normal model. This corrupts the learned normal model and allows the attacker to evade our system. In ensemble-based design, poisoning attacks take a long time and require attackers to make time-sensitive multi-step maneuvers. In our current configuration, the perpetrator must launch a sustained attack for a minimum of 56 hours to corrupt the ensemble. Furthermore, our hash-based distance algorithm makes it hard to simulate packets that contain attack data while resembling normal traffic.

In order to evaluate our system, we conduct experiments using two large data sets collected from a public university's web servers. These datasets contain 1.1 million packets, of which thousands were attack packets. To statistically analyze the data from our experimental results, we manually labeled more than 18,500 alert packets. Our results demonstrate that we are able to achieve a high rate of detection of true attacks while keeping the false alarms very low. More importantly, the reduction in the average number of false alarms, even for traffic collected from a campus with thousands of students, reduces the amount of effort wasted by operators.

The following is a summary of the primary contributions of this work:

- Developed a novel Anomaly Detection based Network Intrusion Detection system for the Web that is based on unsupervised learning and does not require any labeled training data.

- Applied a novel combination of our hash-based distance metric, unsupervised AD system, and ensemble-based techniques to produce outstandingly low false positive rates.

4.2 Related Work

There are many approaches used for network intrusion detection systems (NIDS). In general, systems are categorized into signature-based and anomaly-based systems. Signature-based systems rely on a database of attack signatures and compare new data against these existing signatures. Many signature-based detectors are available commercially (e.g. CISCO [44], Juniper [45], McAfee [46], Snort [42], Suricata [43], Bro [38]). These approaches are widely used, provide high detection rates, and low false positive rates for previously known attacks.

Signature-based systems contrast with anomaly-based sensors, which typically approach the problem by creating a normal model of the system and then detect variations from normal [51]. The advantage of anomaly-based systems is the ability to detect previously unseen (i.e. "zero-day") attacks. However, the resulting algorithms frequently suffer from high false positive rates, making them burdensome for human operators [47,48]. In the following sub sections, we briefly discuss comparable AD sensors and the ensemble techniques used in anomaly detection.

4.2.1 Anomaly Detection

Anomaly detectors designed as NIDS include the Anagram [49] and PAYL [52,53] anomaly detection systems. Anagram stores the n-grams, a contiguous sequence of n characters of a given string, in a Bloom filter [54]. Anagram scores new packets by the percentage of the n-grams not seen in the training data. PAYL models the frequency of 1-grams in the "normal" model and compares the frequency distribution of an incoming packet to the training data. Both these algorithms require clean training data, which places a large burden on the operator, forcing them to label a large number of packets.

The AD sensor we introduce has its roots in existing machine learning principles for transduction [55]. In transduction, potential attacks are determined by comparing the strangeness of incoming packets to the baseline model. The strangeness represents how different an incoming packet is from the normal baseline model. This process is further described in Section 4.3.3. The authors in [50] present DNIDS, a TCM based IDS that uses a strangeness definition taken from the work of TCM classifiers [56]. In [55] an improved strangeness function is introduced, which results in performance improvements of the AD sensor. The KDD 1999 benchmark dataset used to evaluate DNIDS is outdated and known to contain attacks that are easily separable. Therefore the evaluation presented in [56] is unreliable. Additionally, DNIDS yields a 3% FPR, which is an unacceptably high value for a NIDS.

These systems, and the one we present, examine the payload of the packet directly and are known as Content Anomaly Detection (CAD) systems. However, CAD systems do not work for encrypted payloads.

4.3 System Architecture

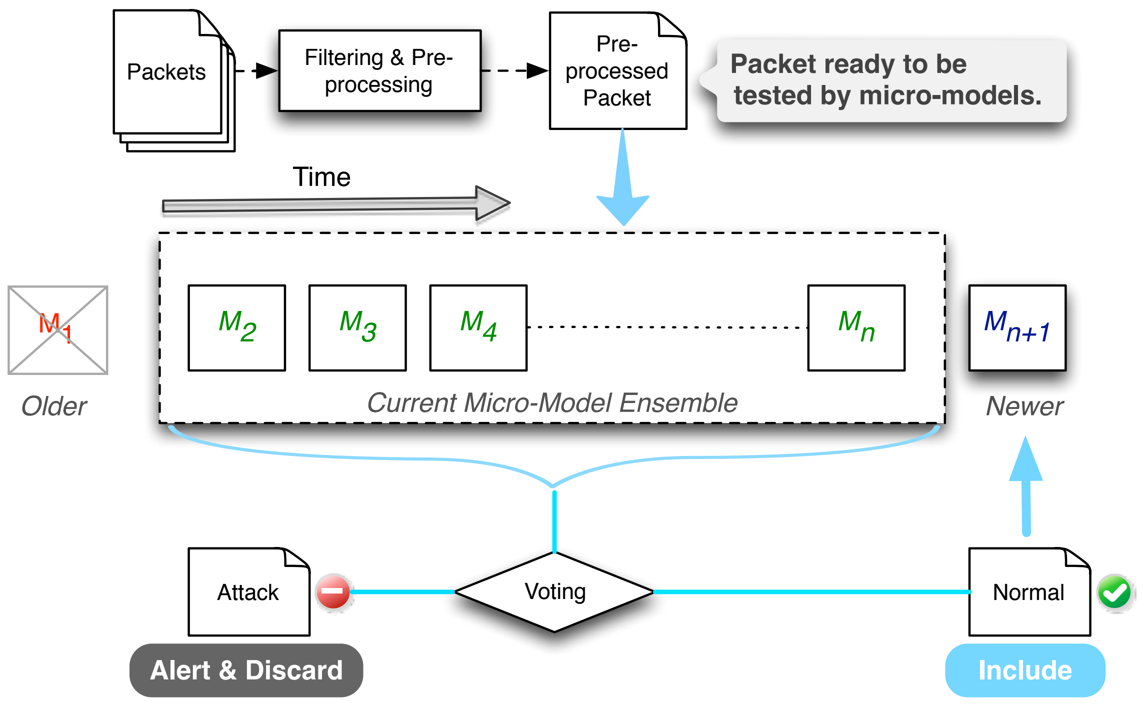


Figure 4.1: transAD Architecture

Our AD sensor consists of the following main components: a filtering and a pre-processing stage, an ensemble of micro-models and a voting mechanism. The system architecture is shown in Figure 4.1. At the filtering stage, only packets of interest are allowed to pass through the filter, while the rest of the packets are discarded. The pre-processing stage extracts the content of the packet and prepares it for use by the AD algorithm. The pre-processed packets are now used to build the ensemble of micro-models. Each micro-model consists of packets collected for a fixed period of time (epoch) called the micro-model duration. Multiple micro-models are created from sequential epochs to form an ensemble. Each micro-model in the ensemble evaluates each test packet, and their decisions are combined using a voting mechanism.

Initially, our AD sensor starts by collecting a baseline of packets that represent the normal incoming pattern of traffic. Before the packets are added to the baseline, packets are filtered and pre-processed. TransAD builds ensembles in real-time and learns the normal traffic pattern by breaking the baseline data into timed "epochs." The baseline data contained in each fixed epoch is referred to as a *micro-model*. Several micro-models are built by collecting packets for multiple epochs to form an ensemble. Once the initial ensemble is ready, new packets are evaluated by the ensemble of micro-models. Every micro-model individually acts as an AD sensor that uses packets contained in the respective micro-model as the normal sample. In order to evaluate if a packet is a potential attack, it is individually evaluated by each micro-model in the ensemble using the *transduction* technique. Individual decisions of each micro-model in the ensemble are combined by a weighted voting scheme to arrive at a final decision. If a packet is voted as an attack, an alert is generated by our sensor. Alerts are then manually inspected by the NIDS operator.

The final component of our sensor helps keep our sensor up-to-date with respect to the changes in the monitored environment over time. The normal traffic patterns may change due to changes in the types of services offered and/or changes in the behavior of users. If the system does not adapt to these changes, the detection and false positive rate of the system may be adversely affected. Our sensor adapts to these changes by building new micro-models using tested packets. Test packets are included in a future micro-model if they are voted as normal. Once newer micro-models are ready, they are incorporated into the ensemble and the older micro-models are discarded.

The current implementation of the system is in c++ and uses *libpcap* [1] and *boost c++ libraries* [2]. We currently process the packets stored in the *tcpdump* format offline faster than real-time. However, our implementation provides hooks to extend this to run on live captures.

[1] `http://www.tcpdump.org/`
[2] `http://www.boost.org/`

4.3.1 Bootstrapping transAD

Bootstrapping is the process of building the initial ensemble of micro-models of our AD sensor. Our sensor targets web traffic containing GET requests because they are a common attack vector used by perpetrators. A GET request method is a commonly used request method of HTTP protocol to communicate with a web server. Our system inspects the contents of the GET request parameters to identify potential attacks. All the GET request packets received by transAD are pre-processed using a filtering and normalization process to enable the AD algorithm to easily identify potential attacks. Next, these packets are used to build the initial ensemble of micro-models. This is followed by a sanitization process where the micro-model ensemble self-cleanses and removes potential attacks. This sub-section describes the bootstrapping process in greater detail.

Although, our system uses GET requests for evaluation, it can be extended to work with other protocols. POST requests, another popular HTTP method, consist of approximately 0.025% of the total requests received by our web server. Since this makes up a very small portion of the requests seen by our AD sensor, we leave consideration of POST requests for the future.

Filtering and Normalization of GET requests

As the packets arrive, the filter removes packets other than those containing GET requests with user-supplied arguments. Only GET requests with user-supplied arguments are allowed through the filter because the user arguments typically contain the actual attack.

Once the packets have been filtered, they are normalized. The normalization process involves the following steps: replacing escaped hex characters in GET requests with their respective ASCII characters; discarding numbers in the GET request parameters because strings form the core of a potential attack; converting the characters into lowercase characters to simplify comparison; and finally, GET request parameters that are shorter than 5 characters are filtered out, as these rarely contain attacks [27]. Figure 4.2 shows a GET request before and after the normalization process. Normalizing the GET request parameters helps our AD sensor to better discriminate between normal and abnormal packets [57]. Therefore, the filtering and normalization processes are applied to all the packets in the bootstrap phase and also to the packets that are tested during normal operation.

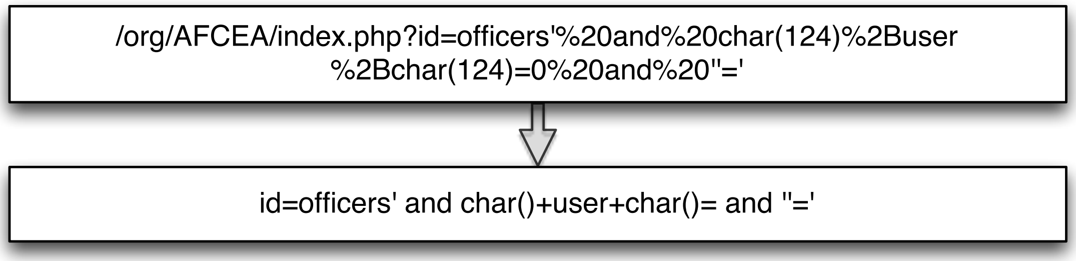

/org/AFCEA/index.php?id=officers'%20and%20char(124)%2Buser %2Bchar(124)=0%20and%20''='
id=officers' and char()+user+char()= and ''='

Figure 4.2: Normalization of an actual GET request.

Building Micro-models

Once the packets are normalized, they are ready to be included in a micro-model. Each micro-model is built using the packets collected for a fixed time duration (epoch) called the micro-model duration. Several micro-models are built using packets collected from consecutive, but disjoint, epochs to form an ensemble of micro-models. Since real network traffic is used to build micro-models, they may potentially contain attack packets.

Most AD sensors are very sensitive to the baseline data used to generate their normal models, and thus require data that is as clean as possible. However, because of the time and effort required to manually label the high volume of traffic processed by NIDS, manual cleansing is not feasible. Based on our experimental data, our sensor appears to be robust to noisy training examples. Even so, the performance of our sensor will improve with clean normal micro-models. In order to ensure we have clean data in our micro-models, we use a sanitization process.

Sanitizing the Micro-models

The sanitization process self-cleanses the initial micro-models without any manual intervention. In the sanitization process, each packet in a micro-model is evaluated by all the other micro-models in the ensemble by using the *transduction* method. The individual decision of the other micro-models in the ensemble are combined using a weighted voting scheme. The weights for each micro-model are in proportion to the number of packets used to build the micro-model. The packets that are voted as potential attacks by the rest of the ensemble are discarded from the micro-model. Only those packets voted as normal are retained in the respective micro-models. The above process is repeated for every packet in every micro-model in the initial ensemble. Thus, the sanitization process self-cleans the micro-model by removing packets that are considered anomalous by a majority of the ensemble.

The sanitization process results in relatively clean data because a majority of the attacks are short-lived, and a given attack is usually limited to a small subset of micro-models. In the sanitization process, a short-lived attack in one micro-model will be voted as abnormal by the majority of the other micro-models in the ensemble. Sanitization removes the offending packet from the micro-model thereby producing a cleaner micro-model. Therefore, the sanitization process self-cleanses the micro-models automatically by using the power of the ensemble. This produces micro-models that closely represent the normal, non-attack network traffic pattern. Sanitization has also shown to improve the performance in comparable systems [2].

At the end of this bootstrap process, we have an ensemble of filtered, pre-processed packets that are included in **sanitized** micro-models that is ready to diagnose future packets.

4.3.2 Model Drift

Once the bootstrapping process is complete, we have a sanitized micro-model ensemble. The sanitized ensemble of micro-models is used to detect potential attacks among incoming packets. However, the initial ensemble of micro-models may become stale over time and may no longer represent the normal traffic pattern. Micro-models become stale for two reasons: first, changes are made to the services offered within a network; second, the behavior of users interacting with the network changes. As a result, if the ensemble is not updated, it may not conform to the normal traffic pattern and AD sensors could produce more false positives. This, in turn, would burden the NIDS operator as time and effort is wasted inspecting false alarms. To adapt to this model drift, we used a scheme where older micro-models are discarded as new micro-models become available.

As packets are being evaluated by the AD sensor, they are used to create new micro-models, using filtered and pre-processed packets voted as normal by the ensemble. When all the packets in an epoch are tested by the ensemble, a new sanitized micro-model is built.

Two drift policies were tested; in the first, the oldest micro-model in the ensemble was discarded as soon as a newer micro-model was available. In the second, the five oldest micro-models were discarded as soon as five newer micro-models became available. Although the drift policy adapts the system to gradual changes over time, drastic changes to the system such as adding a new service will be flagged correctly as potential attacks until the ensemble adapts to the new traffic pattern. Results for each drift policy are presented in the parameter evaluation in Section 4.4.

4.3.3 Distance Metric and Strangeness

To decide if a packet is a potential attack, our sensor utilizes a technique called *transduction* [58]. The *transduction*[3] method computes the fitness of a test packet with respect to a micro-model. The fitness is computed utilizing a function called **strangeness** that

[3]Transduction aims to solve the diagnosing problem for the test packet as opposed to induction, where a general model to diagnose all packets would be built.

measures the uniqueness (or isolation) of the packet. With respect to the normal packets in each micro-model, the fitness test takes the form of a hypothesis test [59], in which the null hypothesis is that "the test packet fits the sample distribution." When the result of the test is statistically significant, we can reject the null hypothesis and therefore consider the test packet a potential attack.

The *strangeness* function can take many forms. In the experiments documented in this paper, we use the sum of the distances to a packet's k-nearest neighbors (k-NN). This strangeness function has shown to work efficiently in [55]. Our sensor uses an improved hash-based distance to measure the similarity between two packets. The greater the distance between two packets, the less similar they are. This hash-distance is used to identify a test packet's k-nearest neighbors in each micro-model; the strangeness of a test packet is the sum of the hash-distance to the k-nearest neighbors.

In order to compute the nearest neighbors to a test packet, we need a suitable distance metric. Since the objective of our implementation is to find abnormal GET request parameter strings, we must use a suitable distance measure that is designed to work with strings. Initially, we tried using Levenshtein string edit distance [60]. When Levenshtein distance was used, the strangeness of a packet was computed after normalizing the distances with the largest distance value observed in a micro-model. However, those experiments resulted in high false positive rates. We solved this problem by introducing a hash-based distance metric that produced a 43% reduction in false positive rates when compared to Levenshtein distance.

The hash distance metric works on n-grams of the normalized GET request parameters. N-grams are sub-sequences of 'n' characters in a string. A sliding window is used to extract all the n-grams from a normalized GET request. A hash table is created with n-grams of the GET request parameters as the key and packet identifier as the value. FNV1a 32-bit [61] hashing algorithm is used to hash the n-grams in the hash table. We chose FNV1a 32-bit hashing algorithm for its efficiency and low collision rate.

To compute the distance between two normalized GET requests, each of the n-grams of the test request is looked up in the hash table. If the hash bucket that contains that n-gram has the identifier of a micro-model request, then an n-gram match is recorded. The similarity metric is computed as a total number of n-gram matches normalized by the number of n-grams in the larger GET request. The similarity metric is now subtracted from 1 to obtain the distance between the two requests as shown in Equation 4.1. For example, in Figure 4.3, the two strings “abcdefg” and “ahbcdz” have one common 3-gram “bcd” that is hashed to the same bucket and is recorded as a match. The number of 3-grams in the larger string is 5. Therefore, the similarity metric is 1/5 and distance between the two strings is 0.8 $(1 - 1/5)$.

$$\text{Distance} = 1 - \frac{\text{total number n-gram matches}}{\text{number of n-grams in the larger string}} \tag{4.1}$$

In the example shown in Figure 4.3, the hash distance performs a simple n-gram match without considering the position of the n-grams in both requests. Simple n-gram matching is a very relaxed measure of similarity and does not take into account the context of the n-grams present in the two requests; it also does not produce a very accurate representation of distance between the two strings. In order to address this issue, we considered the positions of the n-grams in the request while counting the number of matches.

To improve on simple n-gram matching, a relative-distance delta $(r\Delta)$ parameter is introduced to consider a range of possible n-gram positions that should be counted as a match. The simple n-gram matching $(r\Delta = \infty)$ does not consider the n-gram positions. The most restrictive version would require the position of n-grams in the two requests to be the same $(r\Delta = 0)$. If n-gram at position x in the test request has at least one occurrence

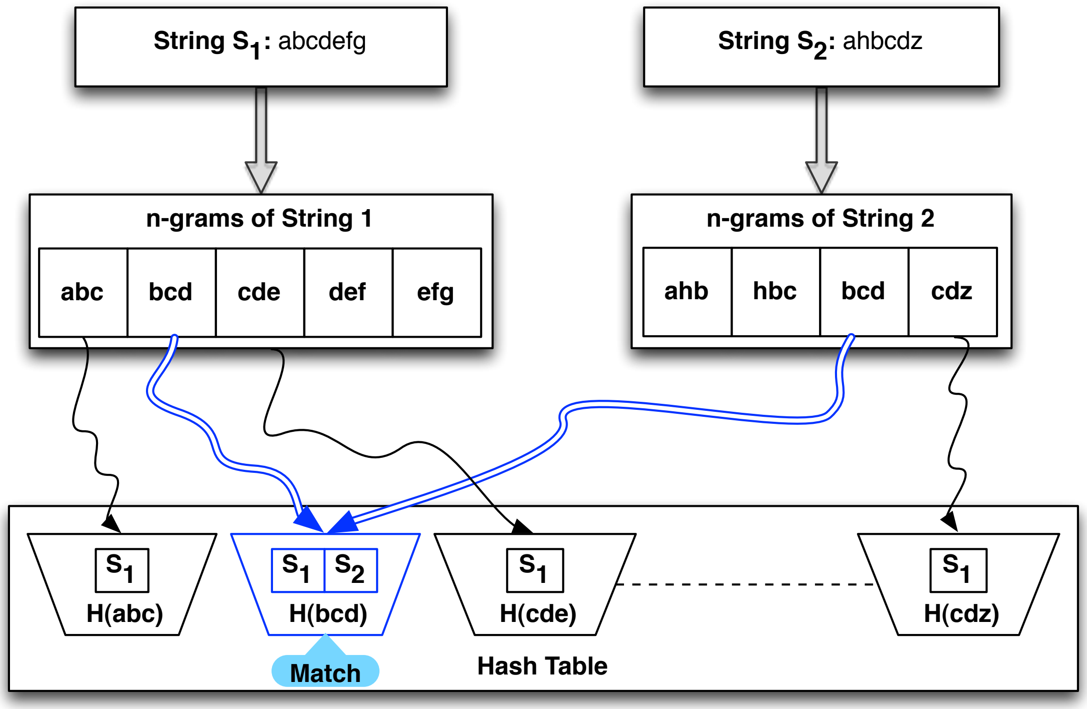


Figure 4.3: Hash Distance: A simple n-gram match is shown between two strings. The hash bucket for the 3-gram 'bcd' contains the identifiers for both strings. Therefore, a 3-gram match is recorded between the two strings. Note: Position of n-grams is not considered in this simple n-gram match approach.

of the same n-gram in the range of positions $[x - r\Delta, x + r\Delta]$ in the micro-model request, the match count is incremented by one. Matches are considered for all the occurrences of the each n-gram in the test request. In order to eliminate the duplicate n-gram matches, the number of matches cannot exceed the occurrence count of the same n-gram in either of the two requests. Therefore, the number of n-gram matches is defined as the minimum of the following values for the n-gram in question:

- the number of matches counted for each occurrence

- the occurrence count in the test request
- the occurrence count in the micro-model request

The algorithm to compute the relative position hash distance is shown in Figure 4.4. The hash distance using the relative position parameter is computed as shown in 4.1. The sum of the hash distances of the k-nearest neighbors is used to compute the strangeness of the test packet, and the fitness test uses the strangeness of the test packet to determine if the packet is a potential attack using a hypothesis test.

```
$matches \leftarrow 0$
$ngramCount1 \leftarrow$ number of n-grams in packet 1
$ngramCount2 \leftarrow$ number of n-grams in packet 2
$maxNgrams = MAX(ngramCount1, ngramCount2)$
for all $ngram$ in packet 1 do
  lookup hash table entry $value$ with the key $ngram$
  if hash table entry $value$ contains ID of packet 2 then
    $tempMatch \leftarrow 0$
    $count1 \leftarrow$ number of occurrences of $ngram$ in packet 1
    $count2 \leftarrow$ number of occurrences of $ngram$ in packet 2
    $minCount \leftarrow MIN(count1, count2)$
    for all $positions1$ of $ngram$ in packet 1 do
      for all $positions2$ of $ngram$ in packet 2 do
        $diff = ABS(position1 - position2)$
        if $diff \leq r\Delta$ then
          $tempMatch \leftarrow tempMatch + 1$
          break
        end if
      end for
    end for
    $match \leftarrow match + MIN(tempMatch, minCount)$
  end if
end for
$distance = 1 - \frac{match}{maxNgrams}$
return $distance$
```

Figure 4.4: Algorithm for computing relative position hash distance.

4.4 Evaluation

In this section we evaluated the performance of our AD sensor using two real-world datasets. We tuned the parameters used by our sensor using the first dataset. Once the parameters were selected, we evaluated the performance of our AD sensor against both datasets. Our results show that using transAD yields high detection rates and low false positive rates. Additionally, we compared the performance of the transAD sensor with another AD sensor, Sanitization Tool for ANomaly Detection (STAND) [2].

4.4.1 Data Sets

We used two datasets [4] consisting of 461 million packets to calibrate and evaluate our AD sensor. The first dataset used to tune and evaluate our system consisted of 13 days of real network traffic arriving at George Mason University's main web server [5] in October 2010. Approximately 223 million packets were captured over this period, of which 25 million packets were HTTP/GET requests. Of the captured HTTP/GET requests, approximately $445,000$ have user arguments that could contain potential attacks. These packets were used to build micro-models and test our *transduction*-based AD Sensor.

A second dataset, which is disjoint from the first dataset, was used to further evaluate our sensor and consisted of real network traffic collected over 14 days in September 2012. The second dataset contains 238 million packets, of which approximately 19 million packets were HTTP/GET requests. This dataset contains approximately $717,000$ HTTP/GET packets with arguments.

In order to evaluate our AD sensor, unique alerts generated by the transAD and STAND for both datasets were manually labeled as attacks or benign packets. To assist in the expert manual inspection of the alert packets, each alert packet's content was compared to attacks seen at honey pot sites and its source IP was checked against black lists and Offensive IP databases. Manually inspecting the contents of all these alert packets was a time-consuming

[4] An IRB approval was received to collect, store, and conduct experiments on this data.

[5] `http://www.gmu.edu/`

process.

The **18,500+** unique alerts generated by both transAD and STAND were manually labeled. Of the total number of unique alerts generated by both transAD and STAND, 13,443 were unique true positive alerts. It was relatively easy to identify true positives, but identifying all of the False Negatives (FNs) was prohibitive. Identifying all of the false negatives would require labeling the complete dataset —an infeasible task with more than 1.1 million packets in our datasets. In order to estimate the false negatives, we identified transAD's false negatives as the true positives identified by STAND but missed by transAD. Similarly, STAND's false negatives were identified as the true positives identified by transAD but missed by STAND. While this method does not give us a perfect count, it was feasible in the time allowed and represents a lower bound on false negatives.

4.4.2 Parameter Evaluation

In order to study the performance of our AD sensor with respect to the parameters described in Table 4.1, we conducted experiments to explore the parameter space. The first dataset with labeled alerts was used to explore the parameter space. Initially, the parameter values were set to the initial values shown in the Table 4.1. The initial values were chosen based on existing literature for comparable AD sensors. Experiments were conducted by varying one parameter at a time while the rest of the parameters were set to initial values. When a better parameter was observed, it was noted as the default value (see Table 4.1). After the default values for all parameters were defined, the experiments were re-run using the default parameters and varying the parameter of interest. The following sub-sections present the results of the effect of each parameter on the performance of our AD sensor.

Micro-model duration (Δ)

Micro-model duration, the size of the epoch chosen for each micro-model, affects the performance of the AD sensor. If the micro-models are constructed for small epochs, they may

Parameter	**Description**	**Initial Value**	**Default Value**
Number of Nearest Neighbors (k)	‘k’ is the number of nearest neighbors used by the k-NN algorithm. Based on the results from [55], we chose 3 for all of our experiments.	3	3
Micro-Model Duration (Δ)	Micro-models used in our sensor are built using packets received in a fixed epoch. The length of this epoch is the duration of each micro-model.	3 hours	4 hours
n-gram Size (g)	The hash-based distance scheme uses n-grams to compute the distance between two GET request parameters. This specifies the number of characters in the n-gram.	6	6
Relative n-Gram Position Matching ($r\Delta$)	For n-grams with matching content, $r\Delta$ specifies the range of positions between the n-grams in the request for them to considered a match.	∞	10
Confidence Level (c)	Confidence level is used by the hypothesis to evaluate if a given packet is normal. Confidence level estimates the reliability of the decisions made by the hypothesis test.	-	80%
Voting Threshold (T)	The voting threshold is the percentage of micro-models in the ensemble that must agree for a packet to be labeled abnormal.	2/3 majority	2/3 majority
Ensemble Size (e)	The number of micro-models used in the ensemble.	25	25
Drift Parameter (r)	The drift parameter denotes the number of old micro-models in the ensemble discarded and the number of staged micro-models inducted into the ensemble at a time.	1	1

Table 4.1: Parameters of transAD

contain only a small sample of packets that do not fully characterize the normal traffic pattern. As a result the k-nearest neighbor algorithm may not be able to find nearest neighbors for normal packets. The inability to find close neighbors may increase the strangeness of normal packets. This may lead to the AD sensor diagnosing normal packets as potential attacks. However, a very large epoch will result in models that are more likely to be contaminated with potential attacks. Such contamination may allow actual attack packets to seem less strange since the distance algorithm may be able to find close neighbors and thus evade our AD sensor. To find the best value for Δ, we fixed all the remaining parameters of the sensor and tested our system for various values of Δ.

Figure 4.5 shows a magnification of the left-hand portion of the Receiver Operating Characteristics (ROC) curve for Δ's between 1 and 5 hours. Each point in a curve is plotted by computing the True Positive Rate (TPR) and FPR at the following confidence levels used for the hypothesis test: 100%, 98%, 95%, 90%, 85%, 80%, 75%, 70%, and 65%. The FPR considerably improves as micro-model duration is increased and reaches a minimum at 4 and 5 hours, and the Area Under the Curve (AUC) reaches a maximum for 4 and 5 hours with the value 0.9989. Also, the graph indicates that our AD sensor has similar performance for Δs of 4 and 5 hours. These results are consistent with the micro-model duration chosen for the Anagram sensor in the sanitization phase [2]. Since there is no additional improvement in the FPR with a 5-hour micro-model duration, we set the default value of Δ to 4 hours.

n-gram Size

Previous work on the Anagram sensor [49] (a predecessor to STAND) studied the distribution of different sizes of n-grams in network traffic and their effect on the AD sensor performance. The authors concluded that higher order n-grams perform well with their sensor. This is due to the higher order n-grams that characterize attacks occurring less frequently in normal packets. The infrequency of the higher order n-grams in normal packets makes it difficult to find nearest neighbors for attack packets. This causes the normal

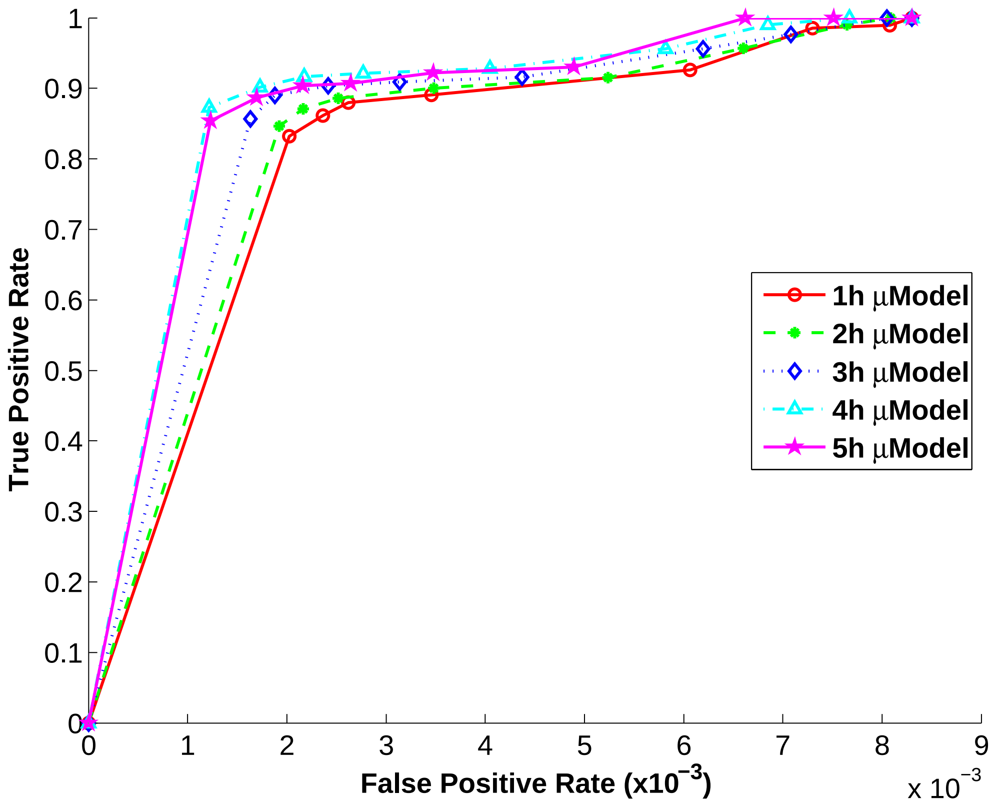


Figure 4.5: Micro-model Duration (Δ): Magnified section of the ROC Curve for transAD at different micro-model durations.

packets to have larger strangeness values.

Figure 4.6 shows the performance of our system with n-gram sizes of 5 through 9. The ROC curve for each n-gram size is plotted for confidence levels 100%, 98%, 95%, 90%, 85%, 80%, 75%, 70%, and 65%. The maximum AUC value of 0.9989 is obtained for n-gram size 6. N-gram size 6 performs the best because n-gram size 5 has a slightly lower detection rate while n-gram size 7 and larger have a higher FPR than n-gram size 6.

The increase in the FPR for sizes 7, 8 and 9 n-grams is due to their low frequency of occurrence. This results in the k-nearest neighbors algorithm not being able to find close neighbors in normal micro-model packets. As a result, some normal packets were strange with respect to the micro-models causing them to be marked as potential attacks. Thus, our results show that as n-gram size increases beyond 6, TPR increases and FPR also increases. The findings for our AD sensor are similar to experimental results using Anagram. We set n-gram size 6 as the default value.

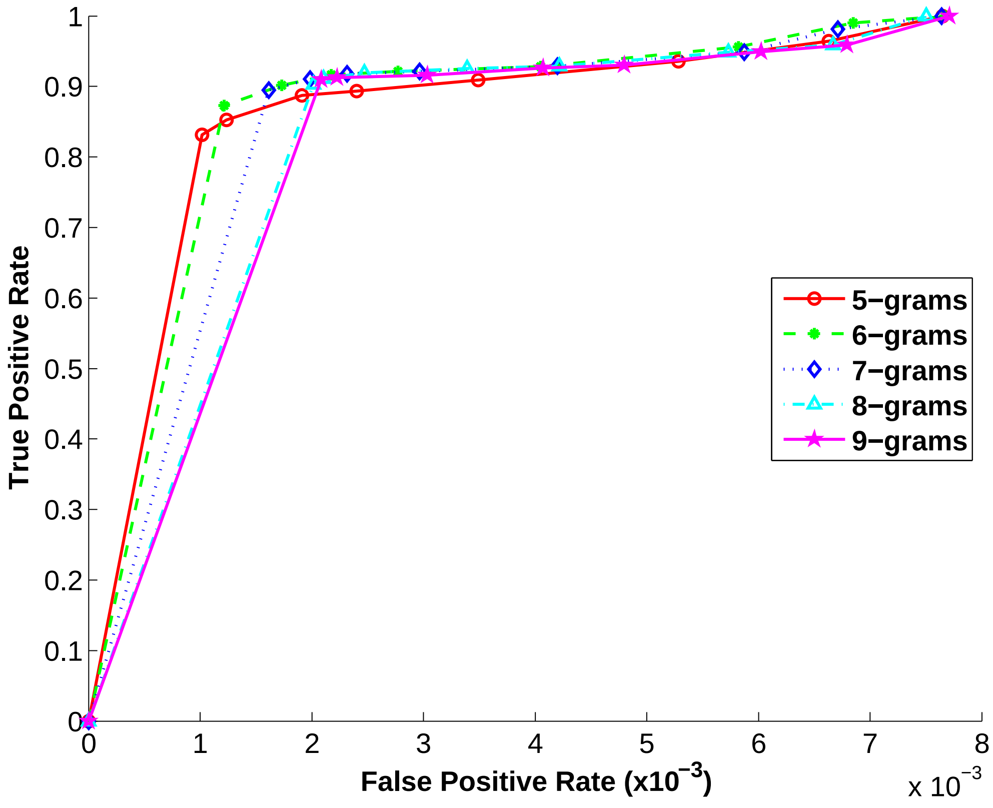


Figure 4.6: n-Gram Size: Magnified section of the ROC Curve for transAD at different n-gram sizes.

Relative Position Matching ($r\Delta$)

The importance of using the relative position parameter for hash distance was introduced in Section 4.3.3. Here, we present the results of using different relative position matching values. Figure 4.7 shows a magnified left-hand portion of the ROC curve for three Relative Position Matching values used in the Hash Distance: exact position matching ($r\Delta = 0$); $r\Delta = 10$; and simple n-gram matching ($r\Delta = \infty$). The ROC curve was plotted by varying the confidence levels as in our other experiments. Simple n-gram matching has the maximum AUC value of 0.999, and $r\Delta = 10$ has the next best AUC of 0.9989. Although $r\Delta = 10$ has a slightly lower AUC compared to simple n-gram matching, we use $r\Delta = 10$ because it improves our system's defense against poisoning attacks. Therefore, we set $r\Delta$ to 10 as the default value.

Voting Threshold

Figure 4.8 shows the magnified left-hand portion of the ROC curves for transAD representing different voting thresholds used to combine the ensembles' decisions. The performance of transAD was tested with the following voting thresholds: 0.67 (absolute majority voting), 0.5 (simple majority), 0.4, and 0.3. The ROC curve was generated by calculating the TPRs and FPRs by varying the confidence values for each threshold, similar to the ROC graphs hitherto presented.

The AUC is at a maximum for the absolute majority voting threshold at 0.9989. Therefore, absolute majority ensemble voting threshold performs better because it has the lowest FPR compared to the other threshold values at similar detection rates. Also, as seen in Figure 4.8, the FPR increases as the voting threshold is relaxed. We use the absolute majority voting threshold as the default value.

Ensemble Size

A very large ensemble size will contain old micro-models that may be stale, which may not as accurately represent the current pattern of traffic. On the other hand, a very small

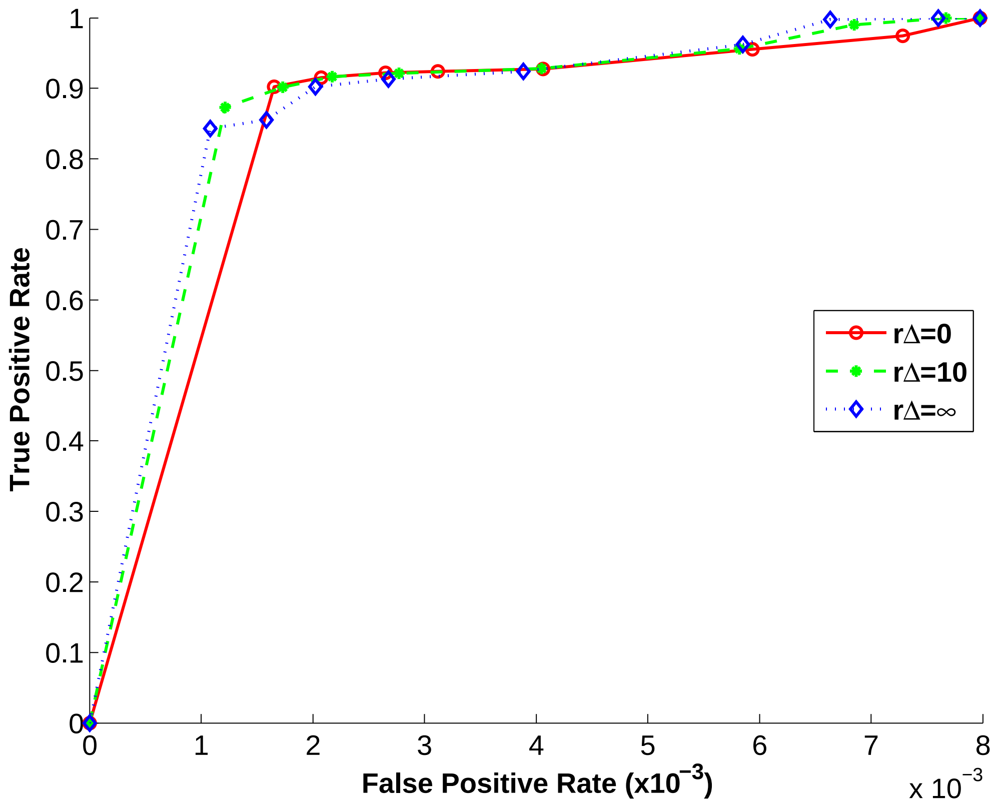


Figure 4.7: Relative Position Matching ($r\Delta$): Magnified section of the ROC Curve for transAD for different relative position matching.

ensemble size might not have enough micro-models to correctly represent the complete normal traffic pattern of the system. Choosing either an overly large ensemble size or an overly small ensemble size could negatively impact the detection rate of the AD sensor.

The effect of varying the number of micro-models in the ensemble is shown in Figure 4.9. This figure shows the section of the ROC curves for ensemble sizes 15, 25, and 35. Ensemble size 25 has the maximum AUC value 0.9989. Hence, ensemble size 25 has a higher detection rate when compared to ensemble size 15 and 35 at the same FPR. Therefore, we chose ensemble size 25 as the default parameter.

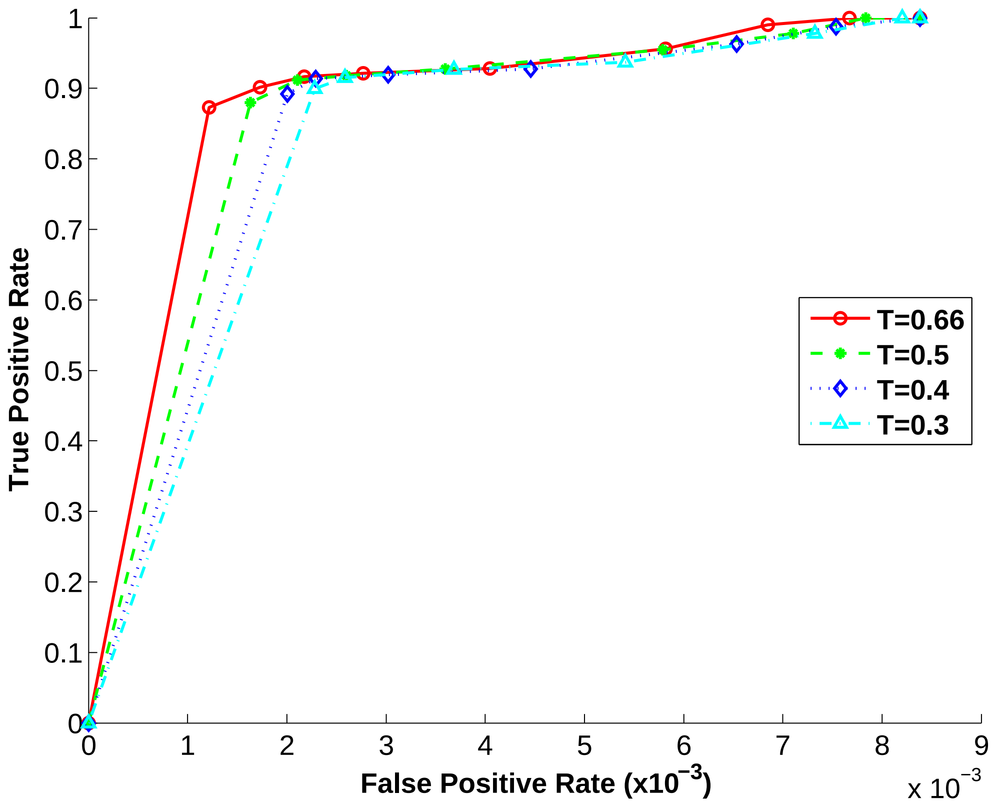


Figure 4.8: Voting Threshold: Magnified section of the ROC Curve for transAD for different voting thresholds.

Model Drift

We examined two different drift settings to adjust for changes in the traffic pattern over time: one where the micro-models were replaced as soon as a new one was available and another where the micro-models were replaced as soon as five new micro-models were available. Figure 4.10 shows the partial ROC curve for drifting micro-models one at a time and five at a time. The AUC for both the drift polices have the same value of 0.9989. Therefore, drift policies tested in our experiments did not show any significant differences. Our results show that updating the micro-models immediately when a new one is ready does not negatively

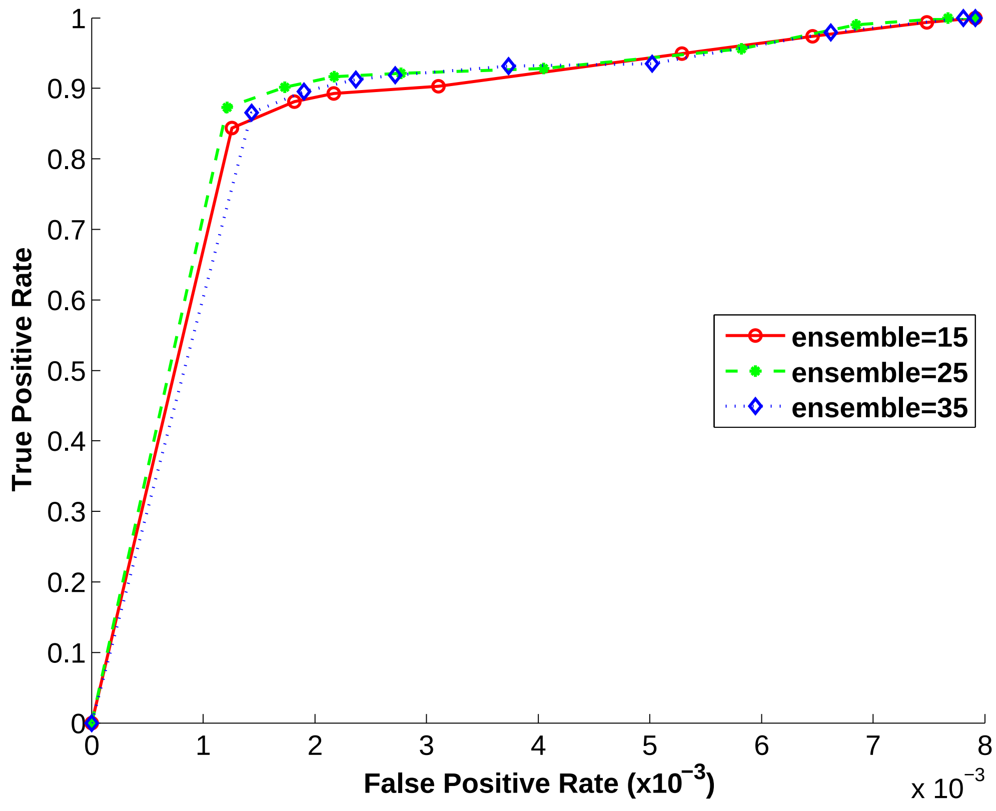


Figure 4.9: Ensemble Size: Magnified section of the ROC Curve for transAD for different ensemble sizes.

impact the detection or false positive rates of the sensor. We therefore, select the drift parameter 1 as the default parameter.

While it is possible to tune the parameters for an individual network as we did, in the future we plan to evaluate our sensor across different networks with the goal of automatically tuning parameters to observed traffic.

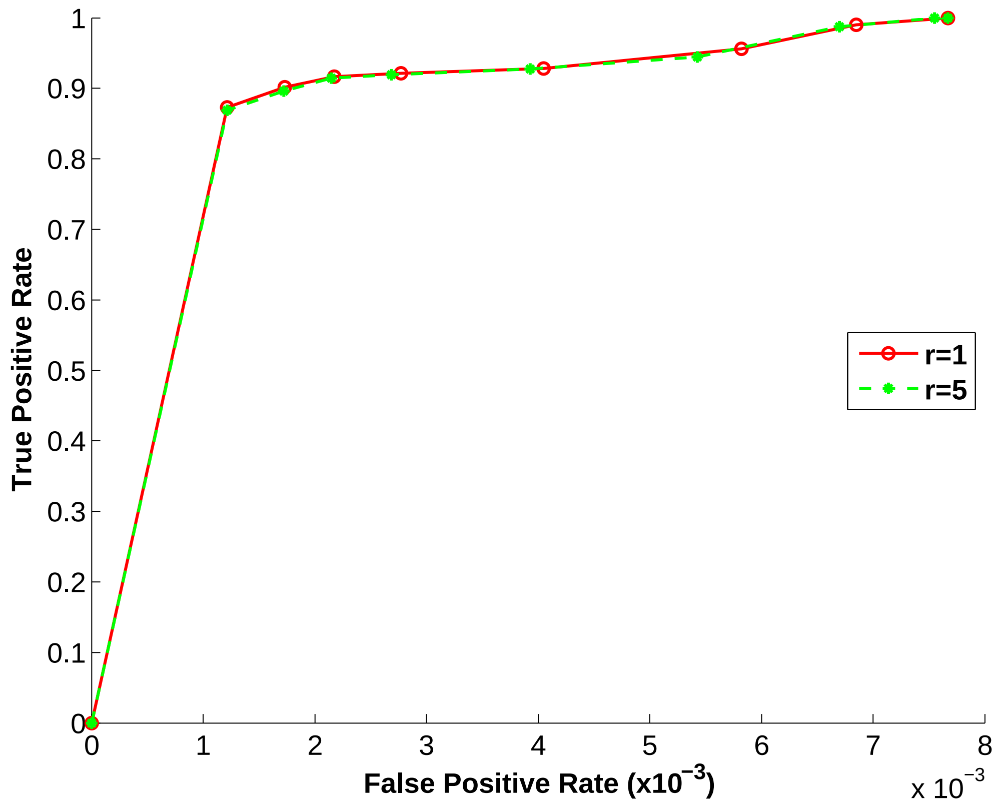


Figure 4.10: Drift Policy: Section of the ROC Curve for transAD for different drift policies

4.4.3 Effectiveness of transAD Against Attacks

This sub-section describes the potential and actual attacks detected by transAD and detection and false positive rates achieved by transAD on the two real-world datasets. Additionally, we discuss the resistance offered by transAD to poisoning attacks.

Types of Attacks Detected

TransAD detected numerous potential attacks in both datasets. None of the potential attacks detected by our sensor required a signature. Since transAD is a self-learning sensor,

Buffer Overflow	/?slide=kashdan?slide=pawloski?slide=ascoli?slide=shukla?slide=kabbani?slide=ascoli?slide=proteomics?slide=shukla?slide=shukla
Remote File Inclusion	//forum/adminLogin.php?config[forum_installed]=http://www.steelcitygray.com/auction/uploaded/golput/ID-RFI.txt??
Directory Traversal	/resources/index.php?con=/../../../../../../../../etc/passwd
Code Injection	//resources-template.php?id=38-999.9+union+select+0
Script Attacks	/.well-known/autoconfig/mail/config-v1.1.xml?emailaddress=********%40****************

Table 4.2: Attacks detected by transAD in the first dataset. Note: The GET request for the script attack has been scrubbed to protect individuals' privacy.

it has the ability to detect never-before-seen attacks without having the need to know any signatures. TransAD did not have any prior knowledge of the actual attacks detected. Therefore, all potential attacks detected by transAD are new. Hence, transAD has the ability to detect other zero-day attacks.

Upon manually examining these potential attacks, many different classes of real attacks were observed. These attacks fall into the following broad classes: buffer overflow, remote file inclusion, directory traversal, code injection, and script attacks. Examples of the type of alerts generated by transAD are shown in Table 4.2. The types of attacks detected depend on the dataset, and SQL injection attacks and buffer overflow attacks occurred more frequently than the rest in our datasets. All of the attacks detected by transAD in the two datasets are real attacks as no artificial attacks were injected.

Detection and False Positive Rates

TransAD was run on the two datasets with an 80% confidence level and the default parameters chosen in Section 4.4.2. True positive, false positive, true negative, and false negative counts and rates are shown for the two datasets in Tables 4.3 and 4.4. The TPR (detection rate) and FPR computed for the first dataset are **92.78%** and **0.40%**, respectively. The TPR and the FPR for the second dataset are **94.77%** and **0.15%**, respectively. TransAD consistently maintains a high detection rate and a very low FPR across both datasets.

Analysis of the data indicates that most of the false positive alerts generated by transAD were related to an online help chat service provided by the technology support center at George Mason University. The online chat support service received queries using GET requests from the campus community. Since the queries asked by individuals were random, the n-grams from these requests did not occur in enough micro-models to be considered normal. This led to these packets being marked as alerts. Other sources of false positives include search queries from the university's catalog and cultural activities page.

Resistance to Poisoning Attacks

In addition to achieving high detection rates and low false positive rates, our AD sensor is not easily susceptible to poisoning attacks. Poisoning attacks, also known as adaptive learning attacks, are where the perpetrator attempts to corrupt the normal model learned by the AD sensor with sustained attacks [49]. In order to successfully poison our AD sensor, the attacker must perform numerous time-sensitive steps continuously over a long period. One starts by sampling and analyzing our traffic pattern. They must then generate an attack packet using a polymorphic attack generator. Once they have generated attack packets, they need to select those packets that resemble our traffic pattern and may evade our AD sensor. Additionally, in order to poison each micro-model, the attacker needs to know the micro-model duration (epoch). Attackers must repeat the above process quickly for consecutive epochs to poison a majority of the micro-model to successfully evade our AD sensor. Performing all of the above steps in a time-sensitive manner over a long period makes poisoning our AD sensor very difficult.

In order to poison a single micro-model, the attacker has to craft seed packets that have contents very close to the normal traffic pattern while containing elements of an attack. To evade our system, these seed packets need to have n-grams that are close to the normal traffic pattern. This would force the seed packet to find close neighbors and be considered normal by our sensor. Therefore in order to construct a successful seed packet, the perpetrator needs know the normal traffic pattern used for that micro-model and the n-gram size. N-gram size used by the sensor is not publicly available to the attacker and makes it hard for the attacker to generate seed packets. If an attacker successfully evades a single micro-model with a single seed packet, they need to repeat the process multiple times with seeds that are incrementally close to the final attack.

In addition to the above challenges, the hash-based distance metric used in our sensor raises the bar for attackers and makes generating an evasive attack packet mimicking normal traffic very difficult. Instead of simple n-gram matching, the hash-based distance metric proposed in this paper takes into account the positions of the n-grams in a packet. The relative distance parameter introduced in Section 4.3 considers n-grams to be a match only when they are positioned within the specified range in the two GET requests. If the the n-gram in the test GET request is not present in the specified range of positions in the micro-model request, it is not considered a match. This increases the distance between the packets and consequently increases their strangeness, and these packets are more likely to be voted as potential attacks. Therefore, the hash-based distance with a relative distance parameter further restricts the attacker's ability to generate a useful attack by limiting the positions and combinations of n-grams in a packet that can be used to generate successful seed attack packets that resembles normal traffic.

TransAD's ensemble-based architecture offers an additional line of defense that makes it robust against poisoning attacks. In order to evade our ensemble of micro-models, the attacker must poison a majority of the micro-models. In order to do this, the attacker needs to know the micro-model duration, ensemble size, and drift scheme, none of which is publicly available and must be inferred by the attacker by observing the system. The attacker must successfully collect information about the normal traffic patterns for each micro-model duration, craft seed packets to evade each micro-model, and sustain the attack until a majority of the micro-models have accepted the seed attack packets and the micro-model containing it is included in the ensemble by the drift scheme. A sustained attack is necessary because, even if attack packets are successfully created to poison a single micro-model, evading our sensor is not possible because the majority of the ensemble would vote against it. If 'Δ' is the micro-model duration, at minimum the attacker needs to craft multiple seed packets every Δ hours and sustain the attack for at least $((e/2) * \Delta) + (r * \Delta)$ hours, where 'e' is the ensemble size and 'r' is the drift parameter. In our present configuration, the attacker must successfully repeat this process every four hours and sustain the attack for a minimum of 56 hours to poison our system. The whole process of poisoning our AD sensor may require multiple rounds of seed packets with seeds that incrementally resemble the final attack. This makes poisoning our sensor a time-consuming process.

Overall, in order to poison our system, the attacker must very carefully build seed packets and poison the first micro-model. It is not possible for the attacker to be certain that the seed packet has evaded the system. The attacker can at most remain cautiously optimistic that his seed has not been rejected by a majority of the ensemble that contains a long history of normal traffic. Since the attacker is not certain if the initial seed can successfully evade the system, they would need to repeat the process by injecting seed packets that incrementally contain data closer to the real attack to poison a majority of the micro-models. Finally an actual attack is launched. Poisoning our sensor requires time-sensitive, multi-step maneuvers and requires a sustained attack over a long period of time, making it very challenging for attackers.

4.4.4 Comparison to Sanitization Tool for ANomaly Detection (STAND)

Common Alerts between transAD and STAND

In this sub-section we compared the alerts generated by transAD and STAND. As we did for transAD, we tested the performance of STAND using both datasets and manually labeled the alerts produced by STAND. Figures 4.11(a) & 4.11(b) show all the alerts that are identified by transAD and STAND. The Venn diagrams show the TP and FP alerts that have been identified by both the algorithms and those that have been identified by only

	Packet Count	**Rate**
True Positives	12,056	92.78%
False Positives	1,125	0.40%
True Negatives	258,758	99.56%
False Negatives	938	7.21%

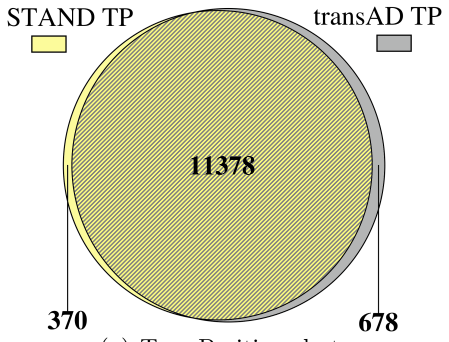
Table 4.3: Results for transAD on the first dataset

	Packet Count	**Rate**
True Positives	41,519	94.77%
False Positives	722	0.15%
True Negatives	251,252	99.71%
False Negatives	2,288	5.22%

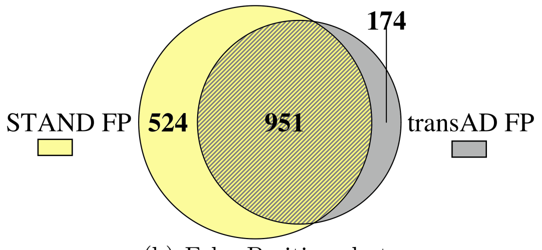
Table 4.4: Results for transAD on the second dataset

one of the two algorithms. In Figure 4.11(a), all occurrences of TP alerts are counted by the AD sensor every time that packet is seen. In Figure 4.11(b), only unique FP alerts are counted. This is because once the operator has inspected and labeled the packet as benign, the AD sensor no longer alerts on that packet. Figures 4.12(a) and 4.12(b) show the Venn diagrams for the second dataset.

Figures 4.11(a) and 4.12(a) show that a majority of the TP alerts are identified by both AD sensors for both datasets. For the first dataset, transAD disjointly identifies 308 more TPs than STAND. These 308 alerts are considered as FN for STAND. Also, transAD identifies 350 fewer disjoint FP alerts than STAND. For the second dataset, transAD disjointly identifies 3,258 more TP alerts and TransAD produces 34 fewer disjoint FP alerts than STAND. TransAD clearly has an advantage over STAND as it produces only a fraction of the FP's compared to the latter. Overall, transAD identifies more TPs and fewer FPs when compared to STAND for both datasets.

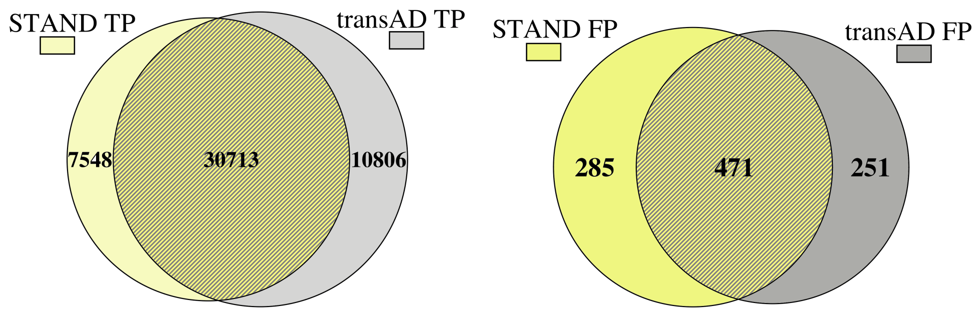


(a) True Positive alerts



(b) False Positive alerts

Figure 4.11: Common TP and FP alerts identified by STAND and transAD on the first dataset.



(a) True Positive alerts (b) False Positive alerts

Figure 4.12: Common TP and FP alerts identified by STAND and transAD on the second dataset.

4.4.5 Comparison Detection and False Positive Rates

In this subsection, we compare the detection and false positive rates of transAD with STAND. The TP, FP, TN, and FN counts and rates generated by STAND for the two datasets is shown Table 4.5 and 4.6. STAND requires a smaller baseline than transAD, hence STAND analyzed more test packets. Therefore, the number of packets tested by the two algorithms are different.

The TPR (detection rate) and False Positive Rate (FPR) for transAD are **92.78%** and **0.40%**, respectively, for the first dataset. As presented in the above tables, the TPR and FPR for STAND are **92.52%** and **0.57%**, respectively. Using transAD reduces the FPR by 29.82%; the TPR increases by 0.28% when compared to STAND.

The TPR and FPR on the second dataset for transAD are **94.77%** and **0.15%**, respectively. The TPR and FPR for STAND are **77.97%** and **0.13%**, respectively. It is interesting to note that for this dataset, STAND has a 14.5% lower detection rate when compared to the first dataset results. STAND does not have consistent detection rates for both datasets. However, transAD performs consistently with over 90% detection rate for both datasets. Using transAD increases the detection rate by 18% when compared to

	Packet Count	**Rate**
True Positives	11,870	92.52%
False Positives	1,475	0.57%
True Negatives	244,278	99.39%
False Negatives	959	7.47%

Table 4.5: Results for STAND on the first dataset.

	Packet Count	**Rate**
True Positives	38,261	77.97%
False Positives	756	0.13%
True Negatives	563,551	99.86%
False Negatives	10,806	22.02%

Table 4.6: Results for STAND on the second dataset.

STAND. The FPR for transAD and STAND were similar.

The two test datasets yielded different results. However, in the first dataset where transAD and STAND had comparable detection rates, transAD performed better with a lower FPR. For the second dataset where transAD and STAND had comparable FPR, transAD performed better with a much higher detection rate. Additionally, transAD consistently maintains high detection rates over both datasets. Overall, transAD performs better across both datasets.

4.5 Conclusions & Future Work

We introduced a new AD sensor, transAD, that combines proven technology with new methods to achieve high detection rates and low false positive rates. Our sensor consistently yielded good results for two real-world datasets. For the two datasets, our AD sensor detected actual attacks, none of which were previously known to transAD, which therefore demonstrated that transAD is able to detect zero-day attacks. Additionally, transAD was shown to be more effective than STAND, a leading AD sensor.

TransAD is a new content-based Anomaly Detection sensor for network intrusion detection based on *transduction*. Our AD sensor uses an unsupervised learning algorithm, which obviates the need for labeled training data. Therefore, our sensor does not require any manual labeling of training data and adapts to changes in the system without additional manual effort.

In addition to requiring no labeled training data, our self-learning sensor uses an ensemble technique of transduction-based AD sensors with our hash-based distance metric to achieve low false positive rates and high detection rates. Also, the ensemble technique and the hash distance metric presented in our paper makes our sensor more robust to poisoning attacks.

We conducted a thorough evaluation of our sensor using two real-world datasets. To validate its performance we manually label more than 18,500 alerts. After tuning the parameters used by transAD, we found our sensor achieves consistently high detection rates for both datasets while having generally lower false positive rates than STAND. This unequivocally shows that our AD sensor, which is a novel combination of transduction, hash-based distance, and ensemble-based techniques, produces very low false positive rates while achieving high detection rates suitable for deployment.

In the future, we propose that the alerts generated by our sensor be processed by a signature-based NIDS. This would help to filter out known attacks and reduce the number of alerts that need to be manually examined by the operator and thus save time and effort. Also, as pointed out by Sommer *et al.* in [47], AD sensors are agnostic to the semantics of the attacks and are unable to help the operator classify the attacks. Combining our system with a signature-based system will help classify already known attacks.

To improve the detection rate, reduce false positive rate, our architecture could be extended in the future to consider reverse neighbors when computing the decisions on packets. A test packet that is normal would have close neighbors in the micro-model. Reverse neighbors involve looking at the converse: how many micro-model packets claim a test packet as a nearest neighbor. A test packet that is normal is expected to be a close neighbor of many other packets in the micro-model. On the other hand, a test packet that is a potential attack may have relatively close neighbors. However, this potential attack packet might not have many close reverse neighbors. Incorporating the concept of reverse neighbors would make it harder to generate seed packets to poisoning attacks and is likely to improve detection and reduce false positive rates.

Finally, in this work, our evaluation is primarily focused on HTTP GET traffic. In the future, we plan to evaluate our sensor with HTTP POST data and other types of network traffic.

Bibliography

Bibliography

[1] Commerce-Land, "History of e-commerce," http://www.ecommerce-land.com/history_ecommerce.html/, Jun. 2004.

[2] G. Cretu, A. Stavrou, M. Locasto, S. Stolfo, and A. Keromytis, "Casting out demons: Sanitizing training data for anomaly sensors," in *Security and Privacy, 2008. SP 2008. IEEE Symposium on*, may 2008, pp. 81 –95.

[3] Y. Song, M. E. Locasto, A. Stavrou, A. D. Keromytis, and S. J. Stolfo, "On the infeasibility of modeling polymorphic shellcode," in *CCS '07: Proceedings of the 14th ACM conference on Computer and communications security.* New York, NY, USA: ACM, 2007, pp. 541–551.

[4] S. Staniford-Chen, S. Cheung, R. Crawford, and M. Dilger, "GrIDS - A Graph Based Intrusion Detection System for Large Networks," in *National Information Computer Security Conference*, Baltimore, MD, 1996.

[5] P. Porras and P. G. Neumann, "EMERALD: Event Monitoring Enabling Responses to Anomalous Live Disturbances," in *National Information Systems Security Conference*, 1997.

[6] F. Cuppens and A. Miege, "Alert Correlation in a Cooperative Intrusion Detection Framework," in *IEEE Security and Privacy*, 2002.

[7] C. Kruegel and T. Toth, "Distributed Pattern for Intrusion Detection," in *Network and Distributed System Security (NDSS)*, 2002.

[8] C. Kruegel, T. Toth, and C. Kerer, "Decentralized Event Correlation for Intrusion Detection," in *International Conference on Information Security and Cryptology*, 2002.

[9] J. Ullrich, "DShield home page," 2005, http://www.dshield.org.

[10] K. G. Anagnostakis, M. B. Greenwald, S. Ioannidis, A. D. Keromytis, and D. Li, "A Cooperative Immunization System for an Untrusting Internet," in *IEEE International Conference on Networks*, 2003.

[11] K. G. Anagnostakis, M. B. Greenwald, S. Ioannidis, and A. D. Keromytis, "Robust Reactions to Potential Day-Zero Worms through Cooperation and Validation," in *Proceedings of the* 9th *Information Security Conference (ISC)*, August/September 2006, pp. 427–442.

[12] V. Yegneswaran, P. Barford, and S. Jha, "Global Intrusion Detection in the DOMINO Overlay System," in *NDSS*, 2004.

[13] A. Farroukh, N. Mukadam, E. Bassil, and I. Elhajj, "Distributed and collaborative intrusion detection systems," in *Communications Workshop, 2008. LCW 2008. IEEE Lebanon*, may 2008, pp. 41 –45.

[14] S. Zaman and F. Karray, "Collaborative architecture for distributed intrusion detection system," in *Computational Intelligence for Security and Defense Applications, 2009. CISDA 2009. IEEE Symposium on*, July 2009, pp. 1 –7.

[15] D. Tian, H. Changzhen, Y. Qi, and W. Jianqiao, "Hierarchical distributed alert correlation model," in *IAS '09: Proceedings of the 2009 Fifth International Conference on Information Assurance and Security.* Washington, DC, USA: IEEE Computer Society, 2009, pp. 765–768.

[16] M. E. Locasto, J. J. Parekh, A. D. Keromytis, and S. J. Stolfo, "Towards Collaborative Security and P2P Intrusion Detection," in *IEEE Information Assurance Workshop*, West Point, NY, 2005.

[17] G. Cretu-Ciocarlie, A. Stavrou, M. Locasto, and S. Stolfo, "Adaptive Anomaly Detection via Self-Calibration and Dynamic Updating," in *Recent Advances in Intrusion Detection.* Springer, 2009, pp. 41–60.

[18] A. Stavrou, G. F. Cretu-Ciocarlie, M. E. Locasto, and S. J. Stolfo, "Keep your friends close: the necessity for updating an anomaly sensor with legitimate environment changes," in *AISec '09: Proceedings of the 2nd ACM workshop on Security and artificial intelligence.* New York, NY, USA: ACM, 2009, pp. 39–46.

[19] B. H. Bloom, "Space/time trade-offs in Hash Coding with Allowable Errors," *Communications of the ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[20] K. Wang, J. J. Parekh, and S. J. Stolfo, "Anagram: A Content Anomaly Detector Resistant to Mimicry Attack," in *Symposium on Recent Advances in Intrusion Detection*, Hamburg, Germany, 2006.

[21] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions and reversals." *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966, doklady Akademii Nauk SSSR, V163 No4 845-848 1965.

[22] J. J. Parekh, K. Wang, and S. J. Stolfo, "Privacy-Preserving Payload-Based Correlation for Accurate Malicious Traffic Detection," in *Large-Scale Attack Detection, Workshop at SIGCOMM*, Pisa, Italy, 2006.

[23] C. Marc-Andre, "IPInfoDB geo-location API," http://ipinfodb.com/ip_location_api.php.

[24] Websense, "LizaMoon," http://community.websense.com/blogs/securitylabs/archive/2011/03/31/update-on-lizamoon-mass-injection.aspx.

[25] G. Vigna, S. Gwalani, K. Srinivasan, E. M. Belding-Royer, and R. A. Kemmerer, "An Intrusion Detection Tool for AODV-based Ad hoc Wireless Networks," in *Computer Security Applications Conference*, 2004.

[26] Y. Song, M. E. Locasto, A. Stavrou, A. D. Keromytis, and S. J. Stolfo, "On the infeasibility of modeling polymorphic shellcode," in *Proceedings of the 14th ACM conference on Computer and communications security*, ser. CCS '07. New York, NY, USA: ACM, 2007, pp. 541–551. [Online]. Available: http://doi.acm.org/10.1145/1315245.1315312

[27] N. Boggs, S. Hiremagalore, A. Stavrou, and S. Stolfo, "Experimental results of cross-site exchange of web content anomaly detector alerts," in *Technologies for Homeland Security (HST), 2010 IEEE International Conference on*, nov. 2010, pp. 8 –14.

[28] A. Lazarevic, A. Ozgur, L. Ertoz, J. Srivastava, and V. Kumar, "A comparative study of anomaly detection schemes in network intrusion detection," in *In Proceedings of the Third SIAM International Conference on Data Mining*, 2003.

[29] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," *Security and Privacy, IEEE Symposium on*, vol. 0, pp. 305–316, 2010.

[30] C. Taylor and C. Gates, "Challenging the Anomaly Detection Paradigm: A Provocative Discussion," in *Proceedings of the* 15^{th} *New Security Paradigms Workshop (NSPW)*, September 2006, pp. xx–yy.

[31] D. Xu and P. Ning, "Privacy-preserving alert correlation: a concept hierarchy based approach," in *Computer Security Applications Conference, 21st Annual*, Dec. 2005, pp. 10 pp. –546.

[32] C. Gates, "Coordinated scan detection," in *Proceedings of the 16th Annual Network and Distributed System Security Symposium (NDSS 09)*, 2009.

[33] R. Sommer and V. Paxson, "Enhancing byte-level network intrusion detection signatures with context," in *CCS '03: Proceedings of the 10th ACM conference on Computer and communications security*. New York, NY, USA: ACM, 2003, pp. 262–271.

[34] M. Norton, D. Roelker, and D. R. S. Inc, "Snort 2.0: High performance multi-rule inspection engine."

[35] G. Vasiliadis, M. Polychronakis, S. Antonatos, E. Markatos, and S. Ioannidis, "Regular expression matching on graphics hardware for intrusion detection," in *Recent Advances in Intrusion Detection*. Springer, 2009, pp. 265–283.

[36] S. Kumar, S. Dharmapurikar, F. Yu, P. Crowley, and J. Turner, "Algorithms to accelerate multiple regular expressions matching for deep packet inspection," in *Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications*. ACM, 2006, pp. 339–350.

[37] P. Lin, Y. Lin, T. Lee, and Y. Lai, "Using string matching for deep packet inspection," *Computer*, vol. 41, no. 4, pp. 23–28, 2008.

[38] V. Paxson, "Bro: a System for Detecting Network Intruders in Real-Time," *Computer Networks*, vol. 31, no. 23-24, pp. 2435–2463, 1999. [Online]. Available: http://www.icir.org/vern/papers/bro-CN99.pdf

[39] Y. Song, A. D. Keromytis, and S. J. Stolfo, "Spectrogram: A mixture-of-markov-chains model for anomaly detection in web traffic," in *NDSS '09: Proceedings of the 16th Annual Network and Distributed System Security Symposium*, 2009.

[40] Symantec, "Internet Security Threat Report, Volume 17," http://www.symantec.com/threatreport/, 2012.

[41] McAfee, "McAfee Threats Report: First Quarter 2012," http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q1-2012.pdf, 2012.

[42] Sourcefire, "Snort intrusion detection system," http://www.snort.org/, Jul. 2012.

[43] Suricata, "Suricata intrusion detection," http://www.openinfosecfoundation.org/, Jul. 2012.

[44] Cisco, "Cisco security products," http://www.cisco.com/en/US/products/hw/vpndevc/products.html/, Jul. 2012.

[45] Juniper, "Juniper network security products," http://www.juniper.net/us/en/products-services/security/, Jul. 2012.

[46] McAfee, "Mcafee network intrusion prevention," http://www.mcafee.com/us/products/network-security/network-intrusion-prevention.aspx/, Jul. 2012.

[47] R. Sommer and V. Paxson, "Outside the closed world: On using machine learning for network intrusion detection," in *Security and Privacy (SP), 2010 IEEE Symposium on*, may 2010, pp. 305–316.

[48] A. Patcha and J.-M. Park, "An overview of anomaly detection techniques: Existing solutions and latest technological trends," *Computer Networks*, vol. 51, no. 12, pp. 3448–3470, 2007. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S138912860700062X

[49] K. Wang, J. Parekh, and S. Stolfo, "Anagram: A content anomaly detector resistant to mimicry attack," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, D. Zamboni and C. Kruegel, Eds. Springer Berlin / Heidelberg, 2006, vol. 4219, pp. 226–248.

[50] L. Kuang, "Dnids: A dependable network intrusion detection system using the csi-knn algorithm," *Queen's University*, 2007.

[51] D. Denning, "An intrusion-detection model," *Software Engineering, IEEE Transactions on*, vol. SE-13, no. 2, pp. 222–232, Feb. 1987.

[52] K. Wang and S. Stolfo, "Anomalous payload-based network intrusion detection," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, E. Jonsson, A. Valdes, and M. Almgren, Eds. Springer Berlin / Heidelberg, 2004, vol. 3224, pp. 203–222.

[53] K. Wang, G. Cretu, and S. Stolfo, "Anomalous payload-based worm detection and signature generation," in *Recent Advances in Intrusion Detection*, ser. Lecture Notes in Computer Science, A. Valdes and D. Zamboni, Eds. Springer Berlin / Heidelberg, 2006, vol. 3858, pp. 227–246.

[54] B. H. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Commun. ACM*, vol. 13, no. 7, pp. 422–426, Jul. 1970. [Online]. Available: http://doi.acm.org/10.1145/362686.362692

[55] D. Barbará, C. Domeniconi, and J. P. Rogers, "Detecting outliers using transduction and statistical testing," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '06. New York, NY, USA: ACM, 2006, pp. 55–64. [Online]. Available: http://doi.acm.org/10.1145/1150402.1150413

[56] V. Vovk, A. Gammerman, and C. Saunders, "Machine-learning applications of algorithmic randomness," in *Proceedings of the Sixteenth International Conference on Machine Learning (ICML-1999)*, 1999, pp. 444–453.

[57] N. Boggs, S. Hiremagalore, A. Stavrou, and S. J. Stolfo, "Cross-domain collaborative anomaly detection: so far yet so close," in *Proceedings of the 14th international conference on Recent Advances in Intrusion Detection*, ser. RAID'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 142–160. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-23644-0_8

[58] A. Gammerman and V. Vovk, "Prediction algorithms and confidence measures based onalgorithmic randomness theory," *Theoretical Computer Science*, vol. 287, pp. 209–217, 2002.

[59] L. Wasserman, *All of Statistics: A Concise Course in Statistical Inference.* Springer, 2010.

[60] V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," Tech. Rep. 8, 1966.

[61] G. Fowler, L. C. Noll, and P. Vo, "Fowler / Noll / Vo (FNV) Hash," http://isthe.com/chongo/tech/comp/fnv/, 1991.

Curriculum Vitae

Sharath Hiremagalore received his Bachelor of Technology (Honors) in Electronics and Communication Engineering from SASTRA University, India in 2007. He received his Master of Science in Computer Science from George Mason University, Fairfax, Virgina in 2011. His research interests include machine learning, network, and system security.