Spectrogram: A Mixture-of-Markov-Chains Model for Anomaly Detection in Web Traffic

Yingbo Song, Angelos D. Keromytis and Salvatore J. Stolfo
Department of Computer Science
Columbia University
New York, NY, 10027, USA
{yingbo,angelos,sal}@cs.columbia.edu

Abstract

We present `Spectrogram`*, a machine learning based statistical anomaly detection (AD) sensor for defense against web-layer code-injection attacks. These attacks include PHP file inclusion, SQL-injection and cross-site-scripting; memory-layer exploits such as buffer overflows are addressed as well. Statistical AD sensors offer the advantage of being driven by the data that is being protected and not by malcode samples captured in the wild. While models using higher order statistics can often improve accuracy, trade-offs with false-positive rates and model efficiency remain a limiting usability factor. This paper presents a new model and sensor framework that offers a favorable balance under this constraint and demonstrates improvement over some existing approaches.* `Spectrogram` *is a network situated sensor that dynamically assembles packets to reconstruct content flows and learns to recognize legitimate web-layer script input. We describe an efficient model for this task in the form of a mixture of Markov-chains and derive the corresponding training algorithm. Our evaluations show significant detection results on an array of real world web layer attacks, comparing favorably against other AD approaches.*

1 Introduction

Today's internet environment is seeing usability out pace security at unprecedented rates. Web-layer code-injections target web applications and take advantage of programming flaws to manipulate the programs behavior, allowing the attacker to manipulate code and data on the target. Whereas traditional software exploits are often used to compromise specific hosts or to launch worms, web-layer attacks operate under a slightly different paradigm, as recent trends show. While the server is victim of the code injection, the targets often include the viewers of that server as well. Compromised websites often discover embedded malcode silently redirecting their viewers to malicious destinations where they are exposed to further exploits. The most common vectors for this class of attacks can be roughly categorized as: cross-site-scripting (`XSS`), PHP local/remote file-inclusion (`L/RFI`) and `SQL-injection`, all of which target port 80. Since memory-layer overflows against server processes also pass through this port, a successful anomaly detection framework for this protocol set would address a range of security concerns. In this paper, we present `Spectrogram`, a machine learning based sensor designed to construct statistical models for acceptable port-80 input and detect attacks as anomalies.

To put the problem into perspective, a recent SANS survey [26] found that attempted attacks on large web-hosting farms in 2007 numbered in the range from hundreds of thousands to millions *each day*. The same source reports PHP `L/RFI` attacks to have peaked at 120,000 distinct sources in that year with over 4000 unique vulnerabilities discovered. This number is orders of magnitude higher than those seen in shellcode based software exploitation vectors. A Sophos 2008 security report [31] indicates that one compromised website is discovered every five seconds, or roughly 16,000 compromised sites per day. Many instances can be attributed to automated web vulnerability scanners and exploitation engines, which are now widely used. Running `Spectrogram` on our university networks, we discovered between one to ten thousand web-layer code-injection attempts per day, from up to one thousand distinct sources; typically `iframe` and `SQL` injections which try to redirect users to drive-by-download sites and file-stealing attempts. These attacks range from crude to rather sophisticated – in one instance, we noticed an attempt to inject a web-shell, written in 2000 lines of PHP, into our server.

Proper coding has been the best defense against code-injection but relying on every programmer to demonstrate

the required level of scrutiny has never been a fully reliable security practice. Intrusion detection sensors complement this strategy by adding specialized layers of input validation. IDS typically falls into two categories: detecting known malcode or detecting legitimate input. Malcode-signature detection solutions, such as `Snort` [29], are effective at filtering known exploits but in a web environment where hundreds of thousands of unique attacks are generated each day and polymorphism is standard practice, the usefulness of signatures is limited. Recently, anomaly detection approaches showed success by modeling acceptable input using statistical models and detecting exploits as anomalies. These approaches though, with some exceptions, have thus far been limited to network layer, protocol-agnostic modeling which are constrained in scope and vulnerable to packet fragmentation and blending attacks. Unlike shellcode and worm traffic, web-layer injections use higher level interpreted code and do not require corruption of the server's control flow at the memory layer. Web layer exploits are smaller, more dynamic and far less complex than shellcode, making them both easier to write and to disguise, by comparison through various obfuscation techniques. Tools such as Metasploit's eVade O'Matic Module (`VoMM`) [21] and `MPack` [24] are seeing widespread use in the obfuscation and automation of web-layer attacks. The latter utilizes invisible `iframe` injections and was responsible for the recent hijackings of thousands of domains.

As an AD sensor, `Spectrogram` allows the defender to focus on learning models customized for the protected server, where training data is available, rather than attempting to predict what the attacker is capable of sending. The method with which to model legitimate content, however, remains an open problem. N-gram modeling has been a promising direction, as shown by the work of Wang *et al.* [12] but for larger gram sizes, the problem quickly becomes ill-posed, which means that small deviations in the training data can lead to large deviations in the performance. This is intuitive given that the sample-space grows exponentially, causing under-fitting problems. Another sensor from our lab, `Anagram` [12], compensates by trading some generalization ability with efficient hashes of known legitimate input. This approach is fast and efficient but begins to suffer when the input space becomes highly dynamic, as in the web-layer context. This article presents the derivations for a new machine learning based probabilistic model that offers more flexibility in the model structure that previous AD approaches and offers a more favorable trade-off between accuracy, generalization ability and speed. Our approach is based on modeling higher order collocations with mixtures of Markov-chains and is designed specifically to capture a representation for not only content but also the structure of script argument strings by learning a proper distribution on the overlapping n-grams.

As a full sensor, Spectrogram offers the following benefits:

1. The introduction of a new Markov-chain factorization that makes n-gram modeling with large gram sizes tractable and algorithmically efficient.
2. A model that captures both the higher order statistical structure as well as content of legitimate requests.
3. A network situated posture to allow monitoring of local and remote hosts as well as log files.
4. Utilization of dynamic packet re-assembly, operating at the web layer to see what the target application sees.
5. Being HTTP protocol-aware, adding white-list flexibility and providing resistance against blending.

We evaluated Spectrogram against a range of web attacks and demonstrate strong performance in detecting many real world exploits. Spectrogram achieves a 97% detection rate on all but one attack vector in unbiased datasets, which use unique samples, and achieves false positive (FP) rates at five orders of magnitude less when evaluated on the full datasets. The only dataset that did not yield high detection accuracy was the one which did not require actual malcode, as explained in later sections.

Organization

This paper is structured as follows: section 2 describes related work in this area. Section 3 describes the architectural design of Spectrogram while section 4 describes the derivations for the Markov mixture-model that lay at the core of our classifier. We describe our experiments and results in section 5 and present discussions on usability in section 5.3, followed by concluding remarks in section 6.

2 Related Work

The main lines of research in network IDS include the following: signature based detection which attempt to extract artifacts from malicious code, anomaly detection (AD) which learns models for acceptable input and detect deviations, emulator based execution of the input to detect code, dynamic vulnerability discovery and tainted data-flow analysis techniques which try to detect role-violations such as when network traffic reaches certain restricted areas of memory. Hybrids of these approaches have also been investigated. Other passive defensive measures operate at the application and operating systems layer and include address-space and instruction-set randomization, stack and other memory protection schemes. Spectrogram falls into network layer AD category; by modeling legitimate content

on port 80, it detects both script and memory layer exploit attempts. This section briefly discusses some representative examples from this range of IDS and related security frameworks.

`Snort` [29] is a well known, signature based network IDS sensor that scans through packets and searches for strings known to be associated with malcode. A great deal of research has been focused on the problem of how to generate signatures automatically from available data. Kim *et al.* [13] presented one approach, as did Singh *et al.* who showed how to extract signatures from worm traffic [28]. The `Polygraph` [22] engine presented by Newsome *et al.* focused on discovering salient artifacts within different instances of network based attacks. Other notable works in this area include [18, 36, 35]. Hybrid approaches, as previously mentioned, include `FLIPS` [20] which filters traffic through an instrumented version of the protected application – if an attack is confirmed, the malcode is dissected to dynamically generate a signature. Cui *et al.* introduced `ShieldGen` [7] uses an instrumented host to discover vulnerabilities and automatically generate patches. Statistical content anomaly detection approaches include the `PayL` sensor [34] which models 1-gram distributions for normal traffic and uses the Mahalanobis distance as a metric to score the normality of incoming packets. Anagram [12] stores portions of legitimate traffic in the form of N-grams into an efficient hash maps, allowing it to efficiently detect unfamiliar N-grams in incoming traffic.

Other approaches include treating all data as potentially executable code and dynamically executing them. Abstract Payload Execution (APE) [32] is a representative example of this line of research. `APE` examines network traffic and treats packet content as machine instructions, aiming to identify NOP-sleds which might be indicative of a shellcode payload. Krugel *et al.* [17] detect polymorphic worms by automatically learning a control flow graph for worm binaries. Anagnostakis *et al.* [1] proposed an architecture called a "shadow honeypot" which is an instrumented replica of the host that fully shares state with the production application and receives copies of messages sent to the protected application — messages that a network anomaly detection component deems abnormal. If the shadow confirms the attack, it creates a network filter to drop future instances of that attack as well as provides positive confirmation for an anomaly detector.

The usefulness of signature based defenses are under debate. Recent work calls into question the ultimate utility of exploit-based signatures [33]. Song *et al.* [30] recently presented a study on the efficacy of modern polymorphic techniques and the potential impact on signature and statistical modeling based sensors. An alternative approach based on vulnerability specific protection schemes have be studied [5, 3, 11]. These methods along with dynamic taint analysis [4, 23] explore techniques for defeating exploits despite differences between instances of their encoded form by attempting to recognize the vulnerabilities themselves. Brumley *et al.* [3] supply an initial exploration of some of the theoretical foundations of vulnerability-based signatures. Such signatures help classify an entire set of exploit inputs rather than a particular instance. The previously mentioned approaches are mainly dedicated to defeating memory corruption/code-injection based attacks such as buffer-overflow and heap-corruption. Web attacks, on the other hand, exploit web-layer scripts and do not require corruptions of the server application's control-flow – at least not at the memory layer.

On the offensive side of web-layer security, Metasploit has introduced the eVade O'Matic Module (`VoMM`) [21] which obfuscates web exploits. The features of this polymorphic engine include white-space randomization, string obfuscation/encoding, variables and function randomization and an array of other techniques. Van Gundy *et al.* explored the design of a polymorphic PHP worm [10]. `MPack` [24], sold within the attacker underground, is an automated browser exploitation tool which injects invisible iframes into target servers which in turn silently redirects viewers to exploit-laden websites that customize the attack based on the user's browser. These engines thwart signature based methods by injecting high degrees of obfuscation into the exploit code. On the defensive side, Reis *et al.* introduced `BrowserShield` [25] which extends the earlier work of Shield [33] to browser protection against exploits hidden in dynamic HTML. Wang *et al.* introduced a content abstraction architecture for separation of execution contexts in browsers. These solutions aim at protecting the user from malicious servers. In the reversed role, of protecting the web-server, the previously mentioned PayL [34] and Anagram [12] sensors have been introduced. Spectator [19] was introduced by Livshits *et al.* as a taint-analysis based approach which tags scripts to prevent Javascript cross-site-scripting worms. Kruegel *et al.* also explored a statistical AD framework for web traffic [15, 16] based on modeling inputs. We elaborate on the similarities between our method and the latter work in section 3.

3 Spectrogram

`Spectrogram` examines individual HTTP requests and models the content and structure of script inputs. In some ways it is analogous to an AD version of the `Snort` sensor, focusing on port 80. This layout offers the flexibility needed to monitor both local and remote hosts, as well as multiple hosts simultaneously. Since the data is modeled at the application layer, `Spectrogram` uses dynamic packet re-assembly to reconstruct the content flow as it would be seen by web applications and, at the same time,

(a) http://www.vulnerable.com/retrieve.php?paperID=302
(b) http://www.vulnerable.com/retrieve.php?paperID={${include($bbb)}}{${exit()}}&bbb=http://www.haxx.org/exploit.txt?
(c) http://www.vulnerable.com/retrieve.php?paperID=../../../../etc/passwd
(d) http://www.vulnerable.com/retrieve.php?paperID=<scriptlanguage=javascript>alert("Our website is moving! Please re-login at our new location: www.vulnerable2.com to access the fileserver!");</script>
(e) http://www.vulnerable.com/retrieve.php?paperID=<iframe src="http://www.haxx.org/exploit.html"></iframe>
(f) http://www.vulnerable.com/retrieve.asp?paperID='/**/union/**/select/**/0,concat(username,0x3a,password)/**/from/**/users/*

Figure 1. Exploitation vectors for scripts which do not properly handle input data. (a) A common script. (b) Remote-file inclusion attack, "exploit.txt" is actually another php script which hijacks the execution of "retrieve.php." (c) Local-file inclusion, the attacker grabs the password file. (d) A victim, tricked into clicking such a link, would see a fake alert re-directing him to a phishing site. (e) iframe injection, silent redirection. (f) SQL-injection.

resist tcp-fragmentation attacks. An offline mode which operates on packet-capture (`libpcap`) data and Apache log files is also available. This section describes the nature of the attacks we are concerned with and the mechanics of the sensor. Details regarding the statistical model we use for the classification engine are described in the next section. In brief, `Spectrogram` learns to recognize short strings accurately. Similar to the way `Snort` performs matching with lexical grammars specified by human experts, `Spectrogram` automatically learns a probabilistic representation for legitimate input from the training data.

3.1 Environment and Threat Model

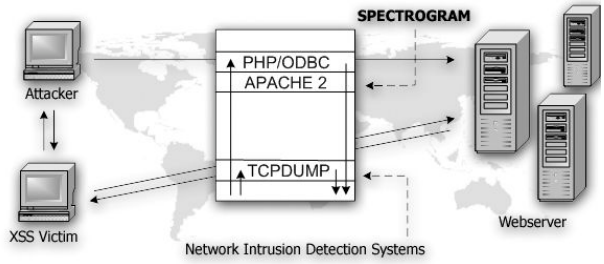


Figure 2. `Spectrogram` is a network situated sensor which operates at the web layer.

Web-layer code-injection attacks manipulate the execution flow of web applications. The attacker takes advantage of programming flaws to inject his own code into the execution context of the target and a successful attack yields a range of results, from data extraction to code execution. As a simple example, consider a script which retrieves PDF files from an archive and writes them to the user while recording the transaction, a common feature of many archival sites. Figure (1) shows some exploits which might succeed if there is no data sanitization (visible parameters in the URL indicate these are `GET` requests). "paperID" is a variable name for this script and takes an integer input value. Without input sanitization, the attacker may inject his own PHP code into the execution context as shown in Figure (1)(b). This attack is known as a "remote file inclusion" and take advantage of PHP's remote library inclusion feature. The attacker may try to steal a file from the server, as shown in Figure (1)(c), in a "local file inclusion" attack. If the server echoes the request back in some unsafe way with an error message such as "paperID: xxx not found" where "xxx" reflects the input then cross-site-scripting (`XSS`) attacks are possible, as shown in Figure (1)(d). This attack tricks other viewers into visiting foreign malicious sites. The XSS attack makes the injected code appear as if it originated from the victim's site. Figure (1)(e) shows the same `XSS` goal accomplished more stealthily with an `iframe` injection. Typically, the foreign site would host additional browser exploits to further compromise the viewers. Finally, Figure (1)(f) Shows a standard SQL-injection that attempts to print the elements of a restricted table. If HTTP `POST` requests were used as the submission method then the XSS victim would never see the injected attack string in the URL as it would remain hidden in the HTTP message body. Memory-layer attacks against the server process itself, such as the `IIS Media` exploit and the `Code Red` worm insert very large strings into the `GET` request field as well. By learning to recognize legitimate strings within HTTP web requests `Spectrogram` attempts to simultaneously address many issues related web-application defense.

3.2 Architecture Design

`Spectrogram` was designed to be a passive IDS sensor. By default, it will only issue alerts for suspected attacks and does not attempt to intercept them. False positives (FP) are a burden on all statistics based IDS solutions and while `Spectrogram` does achieve very low FP rates, as this paper demonstrates, we feel that is most effective when used

as a filter to sieve through potentially millions of web requests (online or offline) in order to find the small subset of interesting attack traffic to bring to a human expert's attention. Such information serves to identify interesting attacks, attack patterns, previously unseen exploits as well as potential vulnerabilities. Being situated at the network layer means that only a port-mirror is needed for the sensor in order to monitor remote hosts or deployment at a proxy junction. Runtime results in this paper illustrate the capacity for real-time detection in such settings. Unlike other NIDS, `Spectrogram` operates *above* the packet layer, at the web/CGI-layer, and was designed specifically to be HTTP-protocol aware. Dynamic re-assembly is used to reconstruct the full HTTP requests as they would be seen by the targeted web application, this yields the side-benefit of resistance against fragmentation attacks since the sensor explicitly reconstruct all fragments. Related sensors [29, 34, 12] which operate at the network layer and perform per-packet inspection can, to certain degrees, be frustrated by evasion tactics such as `overlaying-reassembly` [27] if left in their naive setting. All HTTP requests are parsed to isolate the script argument strings and drop irrelevant content which might skew the anomaly score. Details on the runtime these operations entail are presented in Section 5.

Protocol-aware parsing for HTTP requests is implemented to provide a more detailed level of analysis and to trim out irrelevant features and isolate only the script arguments. Statistical "blending" attacks on AD sensors would be possible if no parsing is used. A trivial example of such an attack is the insertion of legitimate content in the unused protocol fields, this would skew the anomaly score if the entire request is modeled holistically; good content, which does not effect the exploit, is being inserted in addition to malicious code. `Spectrogram`, by default, does not model individual protocol fields such as the `referrer` or `user-agent` strings since these fields have no influence on web-layer code-injection attacks. Software layer attacks against the server applications is an issue but many of these attacks can be detected by the content of the URL strings. If protocol sanitization is required then modeling this field can be turned on as well since these fields accept simple string data as well, though there is arguably little use in modeling such fields for IDS purposes. For further discussion on blending attacks, we refer the reader to Fogla *et al.* [14]. `Spectrogram` examines both `GET` and `POST` requests. For `GET`, we look in the URL to obtain the argument string; for `POST`, the message body. If the server allows input in other ways, such as custom Apache modules, then only new parsers would be needed to extend the framework. The script argument string is also unescaped and normalized before being passed into the AD module for classification. These steps are elaborated in the following section.

`Spectrogram` is built on top of `tcpflow` [9], an efficient content flow re-assembly engine for TCP traffic which utilizes hash tables to reconstruct each flow. The script argument string for each request is extracted for processing — this includes both variable names and corresponding argument values. The combination of the two elements induces a third important feature, characteristic of web-layer requests: *structure*. Script arguments strings within HTTP requests are structured by placing variable name and their respective arguments in pairs, with each pair placed from left to right within the argument string. By examining entire strings, one can capture not only what the contents of the arguments should be but also how they should be positioned. This structure is standard for HTTP and is important as it admits string models for anomaly detection, and is the primary reason why `Spectrogram` utilizes Markov chains. The Markov model we use expands upon earlier works by Kruegel *et al.* [15, 16] whose AD sensor contained a component which modeled content using single-gram transitions and operated on Apache log files. The model we present is a multi-step Markov chain which examines multiple gram-transitions. Furthermore, `Spectrogram` utilizes a mixture of such chains for improved accuracy, designed under machine learning (ML) principles. An ML algorithm which recovers the optimal parameters for this model from the data is offered in this paper, constructed under a supervised learning framework. The previously mentioned sensors [15, 16, 34, 12] use unsupervised algorithms. The distinction is important — while unsupervised approaches eliminates the need for labeled training data, they are guaranteed to find only statistical anomalies — inputs not frequently seen but not necessarily malicious. Trade-offs between FP rates and accuracy are more difficult to address for these sensors and are driven primarily by model complexity. The next section elaborates on these issues and describes the mathematical details of our model.

4 The Spectrogram Model

As previously mentioned, `Spectrogram` is a string model, designed specifically to recognize the distribution of content and structure present within web-layer script input strings. Higher order statistics is used for improved capacity and the ill-posed nature of modeling such short dynamic strings is offset with a Markov-chain relaxation in the dependency assumptions. The Markov-chain structure is appropriate for this problem given the default ordering of content within web-layer inputs. From a statistics perspective, the relaxation induces a more flexible interpolation between the *i.d.* and *i.i.d* extremes that characterize previous models [34, 12]. Figure (3) shows the portion of the request that `Spectrogram` examines — parameter names, their respective inputs, as well as their layout

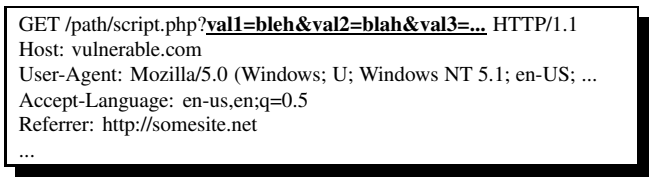


Figure 3. Sample `HTTP` request. Bold, underlined text indicates the portion of the request that is modeled by the sensor. In a GET request, the URL string is parsed; in a POST, the message body.

with respect to each other, are modeled jointly. The sensor is required to infer the likelihood that the input string is legitimate; that not only are the substrings "bleh" and "val1" valid but their order is also valid and that "val2" should follow these substrings. For this purpose, the inference model tracks the n*-gram level transitions* within a string, resolving the likelihood of each observed n-gram given the preceding n-gram: $p(\text{'al1=bleh\&'}|\text{'val1=bleh'})$. Capturing this structure infers that "bleh" is an argument for the variable "val1"; if "val1" is followed by another sequence of unrecognized characters, it would be considered anomalous. The transition-based conditional probability model is analogous to a sliding window that shifts by a single character, with each n-gram sized window dependent upon on the previous. It is intuitive to see then that sufficiently large sized n-grams must be used to obtain an reliable estimate. However, the problem of modeling n-grams is ill-posed as mentioned previously, this means that small deviations in the training data could yield large deviations in the performance. `Spectrogram` compensates by factoring the n-gram distribution into a *product* of $n-1$ pair-wise conditional dependency models. Each character within an n-gram is conditioned on the previous $n-1$ characters. This factorization reduces the problem from exponential to linear complexity and takes advantages of the overlapping nature of the n-grams within an input string — we only need to calculate the likelihood of each character within a string once, though that character may contribute to multiple overlapping n-grams. This explained in more detail in the following section.

Within the `Spectrogram` model, a single Markov chain recovers the likelihood of any given string by calculating the likelihood of each character and then recovering the *geometric mean* of the individual likelihoods. Multiple Markov-chains are used in a linear mixture to obtain the final likelihood score. The only parameters to set within `Spectrogram`'s inference model are the gram size N and the number of mixtures M, these parameters are specified during training. The following sub-sections illustrate the n-gram modeling problem in extended detail and derives the mixture of Markov-chains model step-by-step. In section 4.4 and the appendix, we derive a learning algorithm which automatically learns the model parameters based on the training data.

4.1 N-Grams and the Curse of Dimensionality

N-gram based models have been successfully utilized in recent years for AD roles. Given a string "http://", 2-gram tokens would be "ht", "tt", "tp", *etc*. One seeks to recover an accurate estimation of the distribution of these grams. An example of such a sensor is `Anagram`, introduced by Wang *et al.* [12]. The optimal way to model such a distribution, however, remains an open question. Given an n-gram, if independence between the individual characters is assumed, the sufficient parameters of the model would encompass the frequency of each individual character, independently. This approach requires 256 numbers (valid byte range) and is what the `PayL` [34] sensor uses in its naive setting. If we were to model n-grams jointly to recover an estimate for the distribution of all n-sized token, such as "http://", then estimation of 256^7 individual parameters would be required. In general 256^N numbers are required for gram size N. This approach, to some degree, is what the `Anagram` sensor utilizes. For large n, however, we would never see enough training data to properly fit a full n-gram distribution, making this an *ill-posed* problem. For example, if "val1=AAA&val2=" was seen in the training data but "val1=BBB&val2=" was not, the latter would be flagged as anomalous though it might be a legitimate request. This model suffers from the curse of dimensionality when we attempt to increase its capacity by increasing gram-size. Conversely, when modeling grams independently, as in `PayL`, if three "B"'s were observed anywhere in the training samples then it would be considered normal for them to be anywhere else in the input string, thus throwing away structure information. An effective attack is simply to adding three "B"'s to the end of the attack string to make the request seem more legitimate. `Anagram` addresses the problem with a trade-off between speed and generalization-ability, using hash collisions on subsets of specific n-grams. This allows fast classification and recognition of deviations in static content, such as protocol violation, but can become unstable for small dynamic string content modeling in the web-layer domain, as results in this paper will demonstrate.

In contrast, `Spectrogram` models strings by relaxing the exponentially growing n-gram distribution into an n-step Markov-chain, as an interpolation between the the two previously explored extremes. An n-gram's normality, in this factorization, is conditioned on the $n-1$ preceding grams: given a 5-gram model and input string "http:/", we

condition the normality of the character "/" on the frequency that ":" was observed in the previous position during training, that "p" was observed two positions prior, "t" three positions prior *etc*. Upon examining "val1=BBB&val2=", "BBB" is unrecognized but "val1=" and "&val2=" are recognized. Moreover, they are recognized to be in the correct positions with respect to each other, thus the string appears only *slightly* anomalous due to "BBB"'s presence, as desired. `Spectrogram` also detects padding by immediately flagging strings larger than three standard deviations above the average input length. This acts as a fast ad hoc heuristic filter and should be adjusted per host. These efforts combine to resists statistical blending attacks: if an attacker were to attempt to blend malcode into legitimate traffic, he would need to insert normal content, in the same n-gram distribution as a legitimate request, as well as ensure correct structure, while remaining within the acceptable length, at which point he would be sending a legitimate request and not an attack. Whereas `PayL` requires 256 numbers and `Anagram`, $o(256^N)$, a Markov-chain model requires $256^2 \times (n-1)$ numbers at the n-gram level. Since `Spectrogram` is a mixture of Markov-chains (with mixing weights), $M \times (256^2 \times (n-1)) + M$ numbers are required per model for a mixture of M-Markov chains. M controls the capacity of the model and correlates with the number of *clusters* within the data. Given the dependency structure, the clusters in this case capture the multi-step transitions between alphanumeric characters that encode content and structure, the linkage of certain symbols such as "&", "=", and their overall distributions. In practice, cross-validation can be used to determine the optimal setting for M. The following sub-sections explore the mathematics of this model.

4.2 Factorized N-Gram Markov Models

As previously mentioned, modeling n-grams entails estimating a distribution over an exponentially growing sample-space, making the problem ill-posed. A Markov-chain, on the other hand, leverages the structure of web-requests to reduce the complexity into a linearly growing space. For example, a 2-gram model reduces to a model on 1-gram transitions. Rather than explicitly modeling the likelihood of observing any two characters, the model tracks the likelihood of observing the second character given the first. This conditional model is denoted by $p(x_i|x_{i-1})$, where x_i denotes the i^{th} character within a string and x_{i-1} denotes the $(i-1)^{th}$ character. Extending this concept, the likelihood of an n-gram is driven by the likelihood of x_n and is conditioned on the $n-1$ preceding characters, $p(x_n|x_{n-1}, x_{n-2}, .., x_1)$. The Markov chain approach decouples the preceding $n-1$ characters from each other given the n^{th} character, that is $(x_i \perp x_j | x_n$, where $i, j < n)$, and the joint likelihood is then reset as the *product* of these pair-wise conditionals. A 5-gram model, for example, takes the form $p(x_5|x_4, .., x_1) = p(x_5|x_4)p(x_5|x_3)p(x_5|x_2)p(x_5|x_1)$. We introduce the variable G to indicate the gram size and Equations (1) and (2) shows the interaction of the likelihood values within the larger chain structure:

$$p_G(x_i|x_{i-1}, .., x_{i-G+1}) = \prod_{j=1}^{G-1} p(x_i|x_{i-j}) \tag{1}$$

$$p_G(x_1, ..., x_N) = \prod_{i=G}^{N} \prod_{j=i}^{G-1} p(x_i|x_{i-j}) \tag{2}$$

For the joint likelihood of the entire script argument string, such as the example displayed in Figure (3), we need the product of the individual likelihood values. This is represented in Equation (2) where capital N is used to denote the length of the entire string. The inner product indicates the shifting G-sized window across the larger N-sized string. With this factorization, $n-1$ transition matrices, each of dimensionality 256×256, needs to be kept in memory; this algorithm has complexity growth in $O(n)$. Also notice that since this model is a continuous product of likelihood values, each of which is valued between 0 and 1, it becomes apparent that longer strings will yield lower total likelihoods which is not a desired effect since input length and intent are not strongly coupled. A more appropriate form is a *mean* of likelihood values. The interaction is a product of N values, therefore the N^{th} root is needed, *i.e.* we need to solve for the geometric mean.

$$p_G(x_1, ..., x_N) = \left(\prod_{i=G}^{N} \prod_{j=i}^{G-1} p(x_i|x_{i-j}) \right)^{1/N} \tag{3}$$

Equation (3) calculates the likelihood value for each input string. The capacity of this model can be improved by placing this Markov-chain within a mixture model framework — in the final model, M chains contribute to the final score, with each chain's score weighed appropriately by a scalar mixing coefficient. This model is more general since a mixture model with $M = 1$ represents a single chain. The use of multiple-chains improve upon the capacity of this model by explicitly capturing subclasses of information in a K-means like approach to better capture the potentially many subclasses of input encountered. In fact, the training method we use, `EM`, can be considered as a "'soft"' version of K-means. Since each Markov chain tracks the transitional structure within subclasses of input strings, these clusters correlate more with different types of input structures. For instance, strings with many numerical transitions, strings using many non-alphanumeric characters, *etc*.

4.3 Mixture of Markov Models

Construction of this mixture model follows from standard machine learning procedure of introducing "hidden" states in a weighted-summation mix of all individual chains. Each submodel has the form shown in Equation (3). New input samples would be evaluated over M chains and their values combined in a linear function. Though they share identical structure, these chains have distinct – and independent – model parameters which are recovered from the data. We use θ_i to denote the parameter variable for the i^{th} chain and $\Theta = \{\theta_1, \theta_2, .., \theta_M\}$ to denote a set of parameters for M chains. To clarify, when using models with gram-size G, each θ_i consists of $G-1$ transition matrices. $p(x_i|x_j)$ is the likelihood of a transition from one character to another and is a single value within one of these matrices, indexed by the two characters. The scalar mixing value for a particular chain indexed by s is denoted by π_s. Summing over these submodels with their appropriate mixing weights, $\{\pi_1, \pi_2, , .., \pi_M\}$, yields the final Spectrogram likelihood value:

$$p_G(x_1, .., x_N|\Theta) = \sum_{s=1}^{M} \pi_s \left(\prod_{i=G}^{N} \prod_{j=i}^{G-1} p(x_i|x_{i-j}; \theta_s) \right)^{1/N} \tag{4}$$

Equation (4) represents the M-state mixture model that defines the core classification engine within `Spectrogram`; a subscript G is used to denote a G-gram sliding window. Variable s indicates the hidden state index of the Markov-chains. The mixing proportions all sum to 1: $\sum_{s=1}^{M} \pi_s = 1$. Likewise, the transition likelihoods also sum to 1: $\sum_i p(x_i|x_j) = 1$ for all j, which means that the likelihoods must be normalized.

4.4 Training the `Spectrogram` Model

Training this model means estimating the optimal Θ for Equation (4) and is not as straight forward as training a single Markov-chain since multiple chains are interacting when attemping to fit the data. For a single submodel, the likelihood function is concave in the data and parameters, indicating that a single optimal solution for the parameter setting θ exists. This solution can be recovered by setting the first order derivatives of the likelihood function to zero and solving and resolves to simply counting the number of gram-to-gram transitions within the data then normalizing the matrices so that each row sums to 1. This was done previously for single-gram transition models [15, 16]. The likelihood space for a mixture of Markov-chains, however, is not concave in the parameters and data – a linear mixture of concave functions does not preserve concavity. This removes the single-optimal-solution property and makes solving for the optimal parameter setting more complex. To train a mixture model, we utilize an ML procedure known as Expectation Maximization (EM). The approach utilizes a gradient ascent algorithm to iteratively maximize a lower bound on the likelihood function until no improvement is noted with respect to the estimated parameters.

Let $p(\mathcal{D}|\Theta)$ denote the likelihood of observing a dataset of independent training samples, represented as $\mathcal{D}$. From Bayes Theorem it's known that the optimal setting for parameter Θ is the one that maximizes the joint likelihood of the observation set. If $p(\mathcal{D}|\theta)$ is concave in parameter space then there is a unique optimal solution which can be recovered by solving the gradient of $p(\mathcal{D}|\Theta)$ to zero and solving for Θ. As previously mentioned, for a single Markov-chain, this is simple. $P(\mathcal{D}|\Theta)$ for mixture of Markov-chains, however, is *not* concave in the joint parameter and data space – the hidden states, while increasing model capacity, also remove the concavity property since summations over concave functions do not preserve concavity. This removes the guarantee of the existence of a unique optimal solution. To train this mixture model, we must instead make use of an alternative machine learning procedure known as Expectation Maximization (EM). EM is a popular parameter estimation technique that was first introduced in the simpler form by Dempster *et. al.* [8] in 1977. It is a procedure for optimizing non-concave functions through gradient ascent and contains two core steps: the `Expectation` step (E-step) calculates the joint likelihood of observing the training set given current estimates of model parameters Θ and the `Maximization` step (M-step) finds the gradient of a concave lower-bound on the likelihood function and moves the parameter estimate in that direction. At each steps, the model estimates are *updated* in the direction of the gradient thus maximizing $P(\mathcal{D}|\theta)$ in an iterative procedure. These two steps guarantee monotonic convergence to a local-maximum for $P(\mathcal{D}|\theta)$ and can be alternated until no likelihood-gain is achieved. The EM update rules must be derived on a per model basis. We describe the EM update rules for the `Spectrogram` mixture model below:

E-STEP: In the E-step, we need to solve $P(\mathcal{D}|\Theta)$ for our mixture of Markov-chains model. The likelihood of observing $\mathcal{D}$ is the product of the likelihoods of the individual samples:

$$p_G(\mathcal{D}|\Theta) = \prod_{d=1}^{|\mathcal{D}|} p_G(\mathbf{x}_d|\Theta) \tag{5}$$

The bold faced variable $\mathbf{x}_d$ denotes a string of arbitrary length in $\mathcal{D}$. Next, a lower bound on the expected value is needed. This can be recovered with `Jensen's inequality` which states that given a concave function $f(x)$, we have the identity $f(\sum x) \geq \sum f(x)$. Using $\log$

for $f(x)$, instead of solving for Equation (5) directly, we can solve $\log\left(\prod_{d=1}^{|\mathcal{D}|} p_G(\mathbf{x}_d|\Theta)\right)$. This makes finding the gradient more tractable. Since logarithms are monotonic transformations, the optimal Θ is equivalent for both functions:

$$\arg\max_{\Theta} \log p_G(\mathcal{D}|\Theta) = \arg\max_{\Theta} p_G(\mathcal{D}|\Theta)$$

This means that maximizing the equation in log-space yields the same solution as in the original space. Next, we plug Equation (3) into Equation (5) and solve for the new likelihood function.

$$\log p_G(\mathcal{D}|\Theta) = \log \prod_{d=1}^{|\mathcal{D}|} p_G(\mathbf{x}_d|\Theta) \tag{6}$$

$$= \sum_{d=1}^{|\mathcal{D}|} \log \left(\sum_{s=1}^{M} \pi_s \left(\prod_{i=G}^{N} \prod_{j=i}^{G-1} p(x_i|x_{i-j};\theta_s) \right)^{1/N} \right) \tag{7}$$

$$\geq \sum_{d=1}^{|\mathcal{D}|} \left(\sum_{s=1}^{M} \log \pi_s + \frac{1}{N} \sum_{i=G}^{N} \sum_{j=i}^{G-1} \log p(x_{d,i}|x_{d,i-j};\theta_s) \right) \tag{8}$$

Equation (8) describes the new lower bound on the likelihood functions which we have to maximize. The variable $x_{d,i}$ indicates the i^{th} character of sample string d. To reiterate, $p(x_i|x_j,\theta_s)$ is a single value within the $n-1$ matrices of the s^{th} chain – we are never doing more than retrieving elements from multiple matrices and combining them. With Equation (8), we conclude the derivations for the E-step of the training algorithm.

M-STEP: The maximization step requires solving the gradient of Equation (8) with respect to Θ and the mixing weights $\{\pi_1,..,\pi_M\}$. Given the previously mentioned constraints on the transition matrix, that the rows need to sum to 1, $\sum_i p(x_i|x_j) = 1$ for all j as well as the constraints on the mixing weights $\sum_s \pi_s = 1$, we need to use Lagrange multipliers to find the stationary points under these constraints. For brevity, we provide the final solutions in this paper, the full steps, including further discussions on how to improve the model, will be made available at a later time on our website.[1] The M-STEP proceeds as follows: let $\tau_{d,s}$ denote the log-likelihood of observing string $\mathbf{x}_d$ given model θ_s.

$$\tau_{d,s} = \frac{1}{N} \sum_{i=G}^{N} \sum_{j=i}^{G-1} \log p(x_{d,i}|x_{d,i-j};\theta_s) \tag{9}$$

Each iteration of the EM algorithm shifts Θ in the direction that improves $p(\mathcal{D}|\Theta)$ the most. We used $\pi^\dagger$ to denote how to update the mixing weights and $\theta^\dagger$ for the parameters of the chains.

$$\pi_i^\dagger = \frac{\prod_{d=1}^{|\mathcal{D}|} \pi_i \tau_{d,s}}{\sum_{j=1}^{M} \prod_{d=1}^{|\mathcal{D}|} \pi_j \tau_{d,s}} \tag{10}$$

$$p^\dagger(x_i|x_j;\theta_s) = \frac{p(x_i|x_j,\theta_s) + \sum_{d=1}^{|\mathcal{D}|} \tau_{d,s}}{\sum_{j=1}^{256} \left(p(x_i|x_j,\theta_s) + \sum_{d=1}^{|\mathcal{D}|} \tau_{d,s} \right)} \tag{11}$$

The entire training algorithm for Spectrogram's statistical model is to alternate between these two E and M steps. The training algorithm is given below. Figure (4) shows the pseudo-code for the training algorithm. The algorithm accepts as input, the training dataset $\mathcal{D}$, the gram-size G and the number of Markov-chains M to use and the output is the full mixture of Markov chains model. Recall that G and M control the capacity/power of the model; increasing their values allows the model to fit the data more tightly. The optimal settings should be recovered through cross-validation.

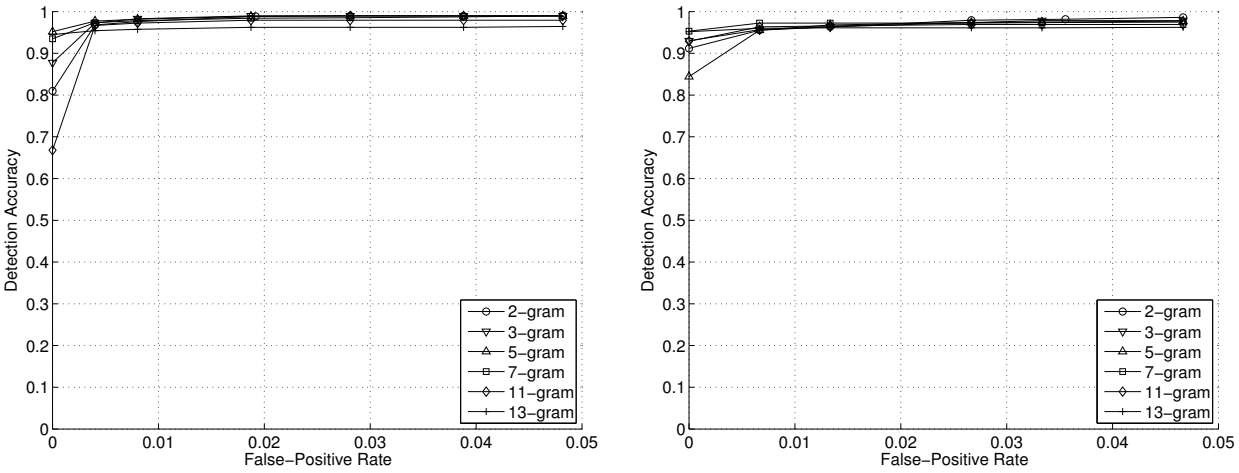
function train-spectrogram $(\mathcal{D}, G, M)$
1 $\{\pi_1,\pi_2,..,\pi_M\}, \Theta = \{\theta_1,..,\theta_M\} \leftarrow$ randomly-initialize
2 $Z_1 \leftarrow$ Equation (8)
3 **for** $i = 2$ **to** ITER-LIMIT
4 Update $\{\pi_1,..,\pi_M\}$ using Equation (9)
5 Update Θ using Equation (10)
6 $Z_i \leftarrow$ Equation (8)
7 **if**$(Z_i - Z_{i-1}) < T$ **then break**
8 **done**
return $\{\pi_1,\pi_2,..,\pi_M,\Theta\}$

Figure 4. `Spectrogram` training with threshold T. The inputs are $\mathcal{D}$ – the dataset, G – the desired gram size and M – the number of Markov-chains to use.

5 Evaluation

We evaluated `Spectrogram` on two of our university's web-servers. One of the machines hosts our departmental homepage and includes scripts for services such as a gateway to a tech-report database, student and faculty directory, search-engines, pages for department hosted conferences, faculty homepages and their accompanying scripts and content that one can typically associate with a computer science department's public facing web server. The second server is a gateway to the homepages of several hundred M.Sc and Ph.D students. We estimate at least several dozen, if not a hundred different scripts running between the two. In our experiments, a single `Spectrogram` mixture-model is trained per server. Two approaches for evaluation

[1] http://www.cs.columbia.edu/ids/



(a) Student server - SQL-Injection.

(b) Dept. server - SQL-Injection.

Figure 5. ROC - `Spectrogram` performance in defending two university servers against SQL injections. In all tables, earlier works [15, 16] are partially represented by the 2-gram model. In this case, small gram sizes worked well given that simple non-obfuscated SQL injection use many non-alphanumeric characters.

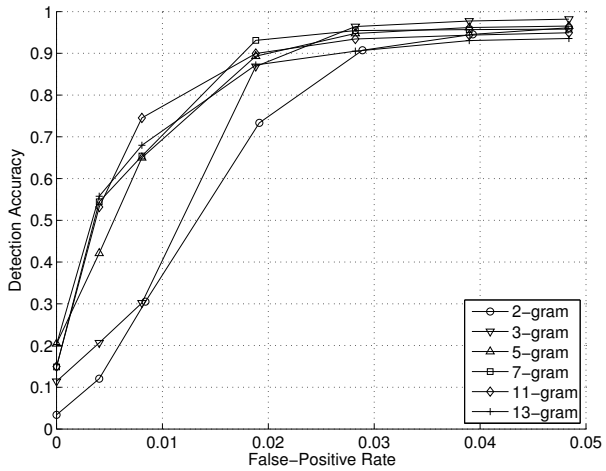
are explored. With **unbiased sampling**, we extract only the *unique* requests within our dataset. This is so we are measuring the capacity of the classifier and are not influenced by the *distribution* of the requests themselves. For example, if a particular request can be classified correctly and that request contributes to the majority of the observed samples, then the FP rate would be biased in our favor and vise-versa. With **full sampling**, we evaluate Spectrogram using the complete dataset of requests seen, giving us a look at the raw FP rate over the entire content stream; all of the attack samples used in our experiments are unique. `Spectrogram` does not use any attack samples in its training; only normal, legitimate input. The detections described in this paper are over completely unseen attack code. When evaluating FP rates, we also use *unseen* legitimate requests given that, with unbiased sampling, each instance of a legitimate request is distinct. We split the dataset into disjoint training and testing sets, thus for normal content, the sets are disjoint as well. Since unique samples are used, the sensor must infer structure and content normality and generalize to unseen samples by looking at subsets of acceptable requests; in order to avoid false positives. Every experiment result reported in this paper is derived from an average of five independent trials where the datasets are completely randomized between tests. For each trial, the dataset of normal requests is randomly split into 95% training and 5% testing disjoint unique sets. All of the attack code is used for each trial.

5.1 Evaluation Dataset

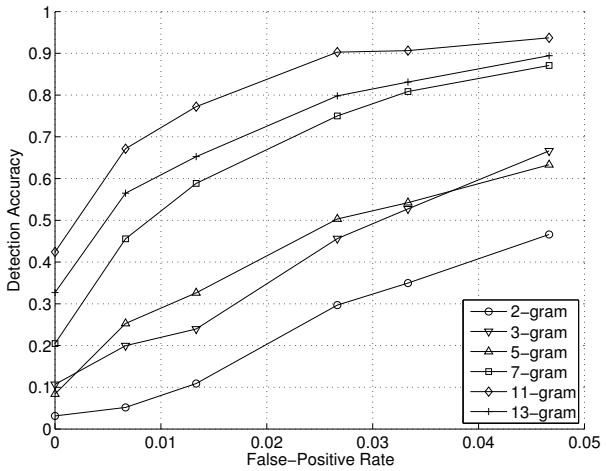
Our dataset includes roughly 6.85 million requests, collected over the period of one month. To generate the training set, we normalized the strings in the manner described in Section (5.3) and extracted only the unique samples. This reduced the dataset to $15,927$ samples for the student server and $3,292$ for the department server. We manually examined the data to ensure that it was free of recognizable attacks (data sanitization is discussed later). The attack dataset included: 637 PHP local and remote file inclusion attacks, 103 Javascript XSS attacks, 309 SQL-injection attacks. We further generated another 2000 unique shellcode samples using four of the strongest polymorphic engines from the `Metasploit` framework, determined using previously published methods [30].

In addition, we passed these samples through `ShellForge`'s [2] ASCII encryption engine to generate 2000 additional samples. Finally, we added four port-80 worms which we had access to: `Code-Red`, `Code-Red II`, `IISMedia` and `IISWebdav`. These worms propagate through the URI and message body and attack web-servers at the memory layer. Roughly half of the samples for `L/RFI` were actual captured attacks against our servers, the same for roughly a quarter of the `SQL-injection` attacks. We collected the remaining samples from sites that host web-exploit code[2]. As previously mentioned, every instance is unique.

[2] milw0rm.com, xssed.com, databasesecurity.com



(a) Student server - File-Inclusion.



(b) Dept. server - File-Inclusion.

Figure 6. ROC - The only poor performance case was the PHP file inclusion attack, due to the fact that this particular class of attack does not require actual malcode, rather only the *address* of the malcode; making them much harder to detect. Notice that increasing gram size makes a big difference.

Attack	(S) PayL	(S) SG-5	(D) PayL	(D) SG-5
L/RFI	5%	**78%**	5%	**74%**
JS XSS	11%	**99%**	9%	**99%**
SQL-Inj.	76%	**98%**	75%	**97%**
Shellcode	100%	100%	100%	100%
ASCII Shc.	100%	100%	100%	100%
Code-Red	√	√	√	√
Code-Red II	√	√	√	√
IIS-Media	√	√	√	√
IIS-Webdav	√	√	√	√

Table 1. Accuracy comparison with FP rate held at 1%. Unique samples used to unbias the data distribution. In all tables, (S) denotes the student server and (D) denotes the department server. Compare with `Anagram` shown in Table (2). See Table (3) for FP rates on the *full* datasets.

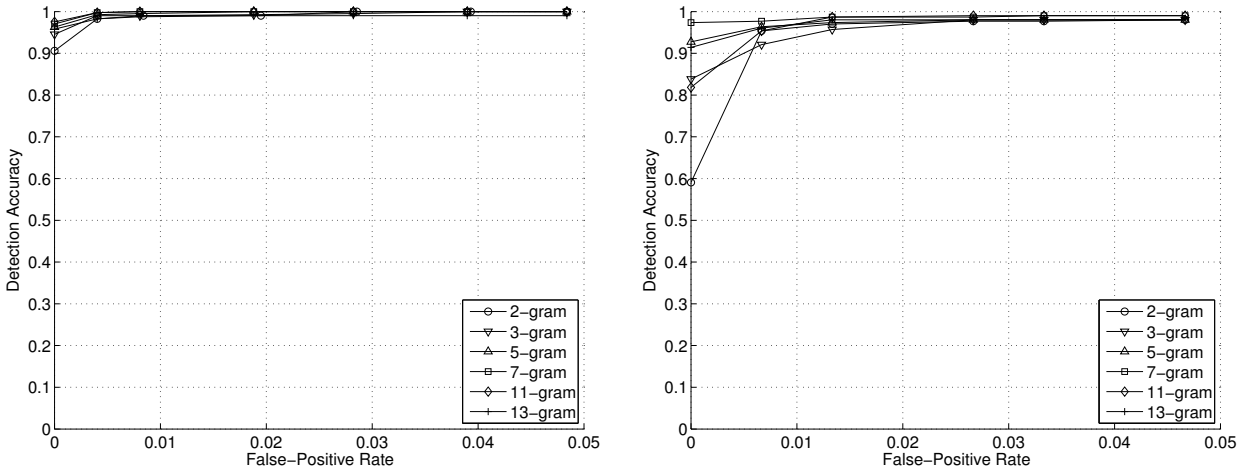
`Anagram`	L/RFI	XSS	SQL	(S) FP	(D) FP
2-Gram	14%	96%	98%	14%	75%
3-Gram	96%	99%	100%	96%	95%
4-Gram	99%	100%	100%	98%	97%
5-Gram	100%	100%	100%	99%	98%

Table 2. Results for `Anagram`; all worms were detected. The high FP is expected since the input is short and dynamic. Compare with `Spectrogram` (`SG-5`) which achieves the same level of detection at 1% FP.

Server	Total Requests	False Positives
Department	2,652,262	118
Student	4,206,176	287

Table 3. FP rates for the full dataset of requests collected over one month. The several orders of magnitude between Tables (1) and (2) (1% vs 0.00006%) is due to the *distribution* of the data. The overwhelming majority of web-requests are easy to classify correctly and using unique samples is needed to accurately evaluate the capacity of the classifier without bias from the sample distribution.

Table (1) shows our accuracy results for `Spectrogram` with a mixture of five Markov chains and gram size 10, abbreviated as `SG-5`. This setting of the sensor was chosen for a good balance in empirical accuracy and performance speed. Our split ratios yields roughly $15,131$ training and 796 distinct testing samples for the student server, and $3,127$ training and 165 testing samples for the department server. In each experiment the datasets are randomized at this ratio and the average of five trials are reported. The `Anagram` results highlight the main problem of underfitting when we increase gram-sizes to increase the power of the classifier. The results show that Anagram is detecting the attacks with higher accuracy when we use larger grams but at the same time it can no longer generalize well. In contrast, `Spectrogram`'s Markov-chain factorization admits



(a) Student server - Javascript XSS.

(b) Dept. server - Javascript XSS.

Figure 7. ROC - Detection of Javascript `XSS`.

much higher gram sizes while maintaining low FP rates on the unbiased dataset. In raw numbers, this 1% false positive rate translated to roughly 8 false positives on average on the student server dataset and for the department server, 2 FPs. The different sensors were all evaluated in the same manner. Good results were noted in all of the experiments except the `L/RFI` attacks. This is because PHP file inclusion attacks do not require actual attack code, just the *address* of the code since the nature of the exploit allows remote code fetching and execution. Due to this fact, detection is much harder but still possible as `Spectrogram`, to some degree, learns that URL inputs into PHP scripts are not legitimate as results show (also c.f. our discussion on evasion tactics). Figures (5,6,7,8) show the ROC curves for `SG-5`, evaluated over a range of attack datasets and gram sizes. These curves demonstrate accuracy for the spectrum of FP rates. Here we can partially compare with the works of Kruegel *et al.* [15, 16]. The 2-gram curves *partially* represent their sensor. We say partially because 2-grams is only one part of their framework. At the same time, our tests used 2-grams in a mixture model while they did not. Their sensor also operates on Apache log files so an exact comparison is not possible. The 2-gram plots are included to drive the point that larger gram-sizes improves performance. Figure (6) and (8) show continued improvement (larger accuracy at the same FP rate) as we increase the gram size. Note that while the FP rate for `Anagram` jumped into the 90's with only 3-grams on the unbiased dataset, `Spectrogram` showed continued improvements in accuracy, in most cases, even beyond 10-grams. In practice, the optimal gram size can be estimated through cross-validation and by generating the same ROC plots as those shown in this paper to find the favorable parameter settings.

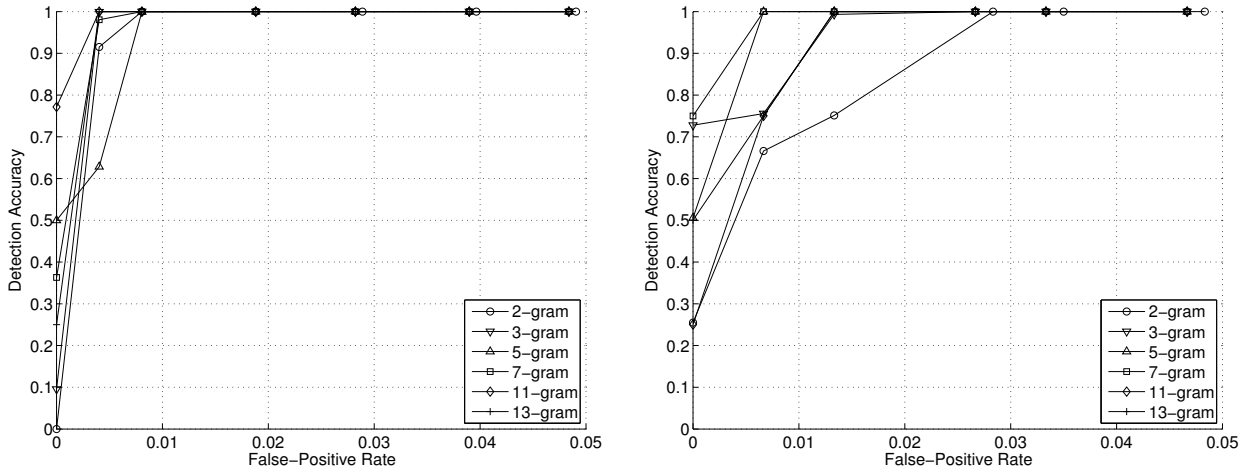
5.2 Runtime

Table 4 shows the run-times for `Spectrogram` at various gram sizes when running on 15,927 samples. Note that training time will depend on the data since we're using a gradient ascent learning algorithm. The convergence rate is a factor and it may take longer with different choices of model parameters, as can be seen from the table – a 2-gram model took longer to train because that the model did not fit the data well and thus the training took longer to stabilize. The cost of reconstructing content flow from network layer packets is greatly reduced with the use of the `tcpflow` [9] library, which is capable of reconstructing nearly 40,000 requests per second on a 3Ghz machine.

5.3 Discussion

Data Normalization: A reduction in the amount of unuseful features within the data is helpful to improve the signal-to-noise ratio. This procedure is mostly ad hoc and should be customized for each server. Methods which we found effective include: un-escaping each string, removing white-space and numbers, and reduction into lowercase. These operations serve to make the input space tighter, by making samples from different sub-classes of legitimate content appear as similar to each other as possible. It also serves to to mitigate the efficacy of some obfuscation methods. Stability is the ultimate goal of this procedure and in deployment, normalization should be tweaked depending on the type of data the monitored web-server(s) observe.

Mixture and gram sizes: As we can see from the ROC curves, there is an obvious benefit to be gained from using larger gram sizes. Clear improvements are observable over the 2-gram model previously studied. The ability to model



(a) Student server - Ascii Shellcode.

(b) Dept. server - Ascii Shellcode.

Figure 8. ROC - ASCII encoded shellcode hide binary `x86` instructions as printable characters and can potentially frustrate statistical sensors. When small gram sizes are used, the sensors lock on only to character distributions and not content. In the small FP range, gram size makes a noticeable difference in discrimination ability.

Gram-Level	Train Time (Matlab)	Sensor Speed (C++)	Model Size
2-Gram	50.3 s	17,094 req/s	3.1 Mb
3-Gram	35.5 s	12,195 req/s	4.6 Mb
5-Gram	54.8 s	7262 req/s	7.7 Mb
7-Gram	69.4 s	4721 req/s	11 Mb
15-Gram	89.8 s	1960 req/s	23 Mb
Request-reassembly	39,000 req/s		

Table 4. Run-times for `SG-5` on a 3Ghz machine (off-line tests). Training is done in MATLAB and the final sensor is implemented in C++. Training time will depend on data and convergence rate. $15,927$ **samples were used and an average of five trials is reported. Packet-reassembly is done using the `tcpflow` library.**

large gram sizes without under-fitting highlights the benefit of the proposed Markov-chain factorization. In practice, the appropriate gram and mixture size will depend on the type of data observed by the monitored server. More dynamic content would require larger mixture sizes while more complex structure/input would require larger gram sizes. The optimal settings for these parameters should be recovered through cross-validation. Methods and tools to automate the cross-validation process are being developed and will be made available. The threshold setting can be automatically adjusted based on the false positive rate on the training data *e.g.* finding the threshold that yields 0.0011% FP.

Training: `Spectrogram` is a supervised learning sensor. In our experiments, we monitored remote hosts and manually labeled legitimate requests. With judicious use of the unix commands: `tr, grep, sort, uniq` and manual examination, we were able to generate a clean dataset of unique requests from over 6 Million samples within a couple of hours. More optimal ways to generate labeled data is under investigation. Since only unique samples of legitimate requests are needed, one possibilility is for the script writers to generate samples of legitimate requests, which they would already do when testing their code. We also refer the reader to the work by Cretu *et al.* [6] on automated data-sanitization and labeling. `Spectrogram`'s model does not admit online/continuous training. However, as we have shown, training using over 15,000 samples requires only a few minutes. Automated nightly or even hourly re-training is not unreasonable to deal with script updates. False positives identified within the logs can be reinserted into the training set so they are recognized in the future. Individual models can also be trained at different

intervals based on the update-frequency of different hosts. Since `Spectrogram` outputs a normality score for each request, it is possible to rank alerts to generate short-lists for human analysis.

Scalability and Forensics: Two main issues concerning scalability are speed and model-capacity. With regard to the former, our results demonstrate that the sensor can perform at sufficient speeds to keep up with thousands of requests per second but it is also possible to deploy several sensors on the network in parallel to monitor different subnets or even individual hosts, as would be the case when protecting large data-centers. Since the sensors are driven by the content of the legitimate data, there is no need to keep individual sensors consistent with each other if the content does not overlap. Further improvement in speed is also possible if implemented on-host, without packet-reassembly. Regarding model-capacity, using larger settings for M and N during training automatically increases the level of details with which `Spectrogram` models the content. This feature addresses highly dynamic content within a specific service. In our experiments using two independent sensors was sufficient for all of our department websites. In practice, if the content stream being modeled is highly dynamic, improved data-normalization procedures can also be introduced to add stability to the model. One potential approach is to incorporate some domain knowledge to filter the content into symbol streams and have `Spectrogram` model operate over these transitions instead.

White-listing: `Spectrogram` is protocol-aware and script-aware and contains a white-listing feature based on the script names and request types. This allows false positive reduction by white-listing scripts with highly dynamic input *e.g.* `POST` with binary content.

Evasion tactics: `Spectrogram` is designed to resist common evasion tactics. Network-layer sensors and certain firewall configurations can be bypassed by fragmentation attacks; `Spectrogram` dynamically re-assembles requests to reconstruct the attack, to see what the target script sees. Polymorphism frustrate signature based sensors; `Spectrogram` utilizes anomaly detection and never trains using malicious content therefore polymorphism has little effect. Some counting-features based statistical AD sensors can have their scoring skewed by attacks crafted to appear like normal requests; `Spectrogram` uses higher order statistics in addition to length. If an attacker were to craft a blending attack to evade this sensor, he would need to insert content and structure that is statistically consistent with normal requests at the n-gram level, while remaining within the acceptable input length, at which point he would be sending a legitimate request. However, if the protected scripts legitimately reads data from foreign sources, with only destination addresses exposed, additional data-sanitization must exist.

6 Conclusions

As the web-exploit threat continues to expand and signature based approaches wane in their usefulness, statistics based IDS solutions which offer more natural resistance against these threats deserve further investigation. `Spectrogram` represents another step in this direction, offering some improvements over previous sensors by designing a model specific for web-layer inputs, as well as an architecture that offers the flexibility needed in a usable NIDS solution, in terms of speed and deployment requirements. This paper studied the AD problem in the context of n-gram modeling, discussed the ill-posed nature of the problem and derived a relaxation of the task into a more tractable linear form through the use of Markov-chains. Our experiments highlight the beneficial effect of these changes. A parameter estimation technique to train this model is offered. Spectrogram has two adjustable parameters: the mixture-size and the gram-size and the optimal settings for both can be found by cross-validation to obtain the desired trade-off between speed and accuracy.

Acknowledgments

This research was sponsored by the NSF through grants CNS-07- 14647 and CNS-04-26623, and by ONR through grant No. N00014-07-1-0907. We authorize the U.S. Government to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of the NSF or the U.S. Government.

References

[1] K. G. Anagnostakis, S. Sidiroglou, P. Akritidis, K. Xinidis, E. Markatos, and A. D. Keromytis". Detecting Targeted Attacks Using Shadow Honeypots. In *"Proceedings of the* 14^{th} *USENIX Security Symposium."*, "August".

[2] P. Biondi". Shellforge Project, "2006".

[3] D. Brumley, J. Newsome, D. Song, H. Wang, and S. Jha". Towards Automatic Generation of Vulnerability-Based Signatures. In *"Proceedings of the IEEE Symposium on Security and Privacy"*.

[4] M. Costa, J. Crowcroft, M. Castro, and A. Rowstron". Vigilante: End-to-End Containment of Internet Worms. In *"Proceedings of the Symposium on Systems and Operating Systems Principles (SOSP)"*, October.

[5] J. R. Crandall, Z. Su, S. F. Wu, and F. T. Chong". On Deriving Unknown Vulnerabilities from Zero-Day Polymorphic and Metamorphic Worm Exploits. In *"Proceedings of the* 12^{th} *ACM Conference on Computer and Communications Security (CCS)"*, "November".

[6] G. F. Cretu, A. Stavrou, M. E. Locasto, S. J. Stolfo, and A. D. Keromytis. Casting out demons: Sanitizing training data for anomaly detection. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)*, May 2008.

[7] W. Cui, M. Peinado, H. J. Wang, and M. E. Locasto". ShieldGen: Automated Data Patch Generation for Unknown Vulnerabilities with Informed Probing. In *"Proceedings of the IEEE Symposium on Security and Privacy"*, "May".

[8] A. Dempster, N. Laird, and D. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society*, pages 1–38, 1977.

[9] J. Elson. tcpflow – A TCP Flow Recorder, 2003. http://www.circlemud.org/ jelson/software/tcpflow/.

[10] M. V. Gundy, D. Balzarotti, and G. Vigna. Catch Me, If You Can: Evading Network Signatures with Web-based Polymorphic Worms. In *Proceedings of the First USENIX Workshop on Offesive Technologies (WOOT)*, August 2007.

[11] A. Joshi, S. T. King, G. W. Dunlap, and P. M. Chen". Detecting Past and Present Intrusions through Vulnerability-Specific Predicates. In *"Proceedings of the Symposium on Systems and Operating Systems Principles (SOSP)"*, October.

[12] S. J. S. Ke Wang, Janak J. Parekh. Anagram: A Content Anomaly Detector Resistant To Mimicry Attack. In *"Proceedings of the* 9^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)"*.

[13] H.-A. Kim and B. Karp". Autograph: Toward Automated, Distributed Worm Signature Detection. In *"Proceedings of the USENIX Security Conference"*.

[14] O. Kolesnikov and W. Lee. Advanced polymorphic worms: Evading ids by blending in with normal traffic. In *Proceedings of the USENIX Security Symposium*, 2006.

[15] C. Kruegel and G. Vigna. Anomaly Detection of Web-Based Attacks. In *Proceedings of the* 10^{th} *ACM Conference on Computer and Communications Security (CCS)*.

[16] C. Kruegel, G. Vigna, and W. Robertson. A Mult-Model Approach to the Detection of Web-based Attacks. *computer Networks*, "48", 2005.

[17] C. Krugel, E. Kirda, D. Mutz, W. Robertson, and G. Vigna". Polymorphic Worm Detection Using Structural Information of Executables. In *"Proceedings of the* 8^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)"*, pages 207–226, "September".

[18] Z. Liang and R. Sekar". Fast and Automated Generation of Attack Signatures: A Basis for Building Self-Protecting Servers. In *"Proceedings of the* 12^{th} *ACM Conference on Computer and Communications Security (CCS)"*, "November".

[19] B. Livshits and W. Cui. Spectator: Detection and containment of javascript worms. In *"Proceedings of the USENIX Annual Technical Conference"*, June 2008.

[20] M. E. Locasto, K. Wang, A. D. Keromytis, and S. J. Stolfo". FLIPS: Hybrid Adaptive Intrusion Prevention. In *"Proceedings of the* 8^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)"*, pages 82–101, "September".

[21] Metasploit Developement Team. Metasploit Project, "2006".

[22] J. Newsome, B. Karp, and D. Song". Polygraph: Automatically Generating Signatures for Polymorphic Worms. In *"Proceedings of the IEEE Symposium on Security and Privacy"*, "May".

[23] J. Newsome and D. Song. Dynamic Taint Analysis for Automatic Detection, Analysis, and Signature Generation of Exploits on Commodity Software. In *"Proceedings of the* 12^{th} *Symposium on Network and Distributed System Security (NDSS)"*, February 2005.

[24] Panda Labs. MPack Uncovered, "2007".

[25] C. Reis, J. Dunagan, H. J. Wang, O. Dubrovsky, and S. Esmeir. Browsershield: Vulnerability-driven filtering of dynamic html. In *Proceedings of Symposium on Operating Systems Design and Implementation (OSDI)*, 2006.

[26] SANS. SANS Top 20. http://www.sans.org/top20/.

[27] S. Siddharth. Evading NIDS, revisited, 2005. http://www.securityfocus.com/infocus/1852.

[28] S. Singh, C. Estan, G. Varghese, and S. Savage". Automated Worm Fingerprinting. In *"Proceedings of Symposium on Operating Systems Design and Implementation (OSDI)"*.

[29] Snort Development Team. Snort Project.

[30] 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 ACM Conference on Computer and Communications Security (CCS)"*.

[31] Sophos. Security threat report. Technical report, Sophos, https://secure.sophos.com/security/whitepapers/sophos-security-report-2008, 7 2008.

[32] T. Toth and C. Kruegel". Accurate Buffer Overflow Detection via Abstract Payload Execution. In *"Proceedings of the* 5^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)"*, pages 274–291, "October".

[33] H. J. Wang, C. Guo, D. R. Simon, and A. Zugenmaier. Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits. In *Proceedings of the ACM SIGCOMM Conference*, pages 193–204, August 2004.

[34] K. Wang, G. Cretu, and S. J. Stolfo". Anomalous Payload-based Worm Detection and Signature Generation. In *"Proceedings of the* 8^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)"*, pages 227–246, "September".

[35] J. Xu, P. Ning, C. Kil, Y. Zhai, and C. Bookholt". Automatic Diagnosis and Response to Memory Corruption Vulnerabilities. In *"Proceedings of the* 12^{th} *ACM Conference on Computer and Communications Security (CCS)"*, "November".

[36] V. Yegneswaran, J. T. Giffin, P. Barford, and S. Jha". An Architecture for Generating Semantics-Aware Signatures. In *"Proceedings of the* 14^{th} *USENIX Security Symposium"*.