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

Abstract. We present `Spectrogram`, a mixture of Markov-chains sensor for anomaly detection (AD) against web-layer (port 80) code-injection attacks such as PHP file inclusion, SQL-injection, cross-site-scripting, as well as memory layer buffer overflows. Port 80 is the gateway to many application level services and a large array of attacks are channeled through this vector, servers cannot easily firewall this port. Signature-based sensors are effective in filtering known exploits but can't detect 0-day vulnerabilities or deal with polymorphism and statistical AD approaches have mostly been limited to network layer, protocol-agnostic modeling, weakening their effectiveness. N-gram based modeling approaches have recently demonstrated success but the ill-posed nature of modeling large grams have thus far prevented exploration of higher order statistical models. In this paper, we provide a solution to this problem based on a factorization into Markov-chains and aim to model higher order structure as well as content for web requests. Spectrogram is implemented in a protocol-aware, passive, network-situated, but CGI-layered, AD architecture and we show in our evaluation that this model demonstrates significant detection results on an array of real world web-layer attacks, achieving at least 97% detection rates on all but one dataset and comparing favorably against other AD sensors.

1 Introduction

Web-layer attacks represent a unique sub-category within the larger code-injection attack vector, given that port 80 is the gateway to many application level services. As internet technology grows, numerous Web 2.0 services are appearing every day, written in a growing array of dynamic environments. Web-portals, Wikis, bulletin boards, photo galleries *etc.* have become ubiquitous and contributes to an ever expanding exploitable vulnerability base. These services share the port 80 as a common interface and, as a result, a multitude of attacks are channelled through this port, making web-layer defense an issue of critical importance. The most troubling and wide-spread of these attacks can be broken into these major categories: cross-site-scripting (`XSS`), local/remote file-inclusion (`L/RFI`) and `SQL-injection`. In addition, traditional memory layer shellcode injection against server applications also aim their attacks at port 80.

SANS reported that attacks on large web-hosting farms in the year 2007 numbered in the range of hundreds of thousands to millions *each day* [28]. The same source reports PHP `L/RFI` attacks to have peaked at 120,000 distinct sources in the year 2007, with over 4000 unique vulnerabilities discovered – a number significantly larger than those encountered in traditional shellcode based software exploitation vectors. Automated web vulnerability scanners are now rampant. Our own university receives between 1000 to ten thousand web-layer attacks, from up to 1000 distinct sources, each day. These numbers come from validated detections using the Spectrogram sensor described in the paper. The payloads of these attacks typically consists of `iframe` and SQL injection to redirect users to drive-by-download sites or local file stealing attempts. We've even noticed an attempt to inject an entire shell interface written in (2000 lines of) PHP into our server, and we suspect many other universities are being similarly targeted.

Due to the volume of traffic seen, the dynamic environment, the high range of sources as well as the use of proxys, it is nearly impossible to write one universal policy to determine acceptable input, therefore few servers firewall port 80. This effectively allows the previously mentioned attacks to pass

through unhindered and reach the target application. In fact, HTTP tunneling [10] was invented in order to allow attackers to pipe other, non web-layer, attacks through this often undefended port. Malcode signature detection approaches, such as Snort, are often effective at detecting known attacks, but are limited because they must know what the attack looks like beforehand. Recently, statistical anomaly-detection approaches have had success by modeling acceptable normal content but these approaches (with some exceptions) have thus far been limited to network layer, protocol-agnostic modeling which are fundamentally constrained in scope and fall vulnerable to packet fragmentation and blending attacks. Unlike the shellcode and worm traffic that these AD sensors were designed to detect, web-layer injections are aimed at CGI-layer exploitation and don't require corruption of the server's control flow at the memory layer. As such, they are far less constrained than shellcode, making them both easier to write and to disguise by comparison. Metasploit's eVade O'Matic Module (VoMM) [23] and tools like MPack [26] are openly distributed tools which obfuscate and automate web-layer attacks. The latter software utilizes invisible `iframe` injections and was responsible for the recent hijackings of thousands of domains on large web-hosting farms. We showed in a recent paper how even shellcode can now be highly obfuscated by modern engines [32], malcode signatures have virtually no hope against script-level attack obfuscation on vectors and payloads that are far less structurally constrained.

Given this, Spectrogram was designed to be an AD sensor, though anomaly detection solutions come with their own set of challenges. For one, the optimal way to model textural content is largely an open problem. N-gram modeling has been a promising direction, as shown by the work of Wang *et. al.* [14] but for larger gram sizes this quickly becomes an *ill-posed* problem, meaning that small deviations in the training sample can lead to large deviations in the performance. This is because the sample space grows exponentially, $O(256^n)$, and under-fitting problems quickly arise. Another sensor from our lab, Anagram [14], tries to deal with this by using hash collisions in Bloomfilters. Unfortunately, modeling large gram sizes ($n > 10$) quickly becomes impossible due to memory requirements and for script input modeling, under-fitting problems show up at smaller gram sizes. This article presents the derivations for a new probabilistic model for n-grams. Our approach is based on modeling higher order collocations with mixtures of Markov-chains and is designed specifically to model not only content but also the *structure* of script argument strings. Implemented as a sensor, Spectrogram's main benefits are that it:

1. Introduces a new Markov-chain factorization that makes n-gram modeling with large gram sizes both tractable and algorithmically efficient.
2. Models both the higher order statistical *structure* as well as content of legitimate requests.
3. Is network situated to allow monitoring of remote as well as local hosts but also has a forensics feature for anomaly detection in Apache logs.
4. Utilizes dynamic packet reassembly and operates at the CGI-layer to see what the target application sees and to counter fragmentation based evasion tactics.
5. Is HTTP protocol-aware to add white-list flexibility and provide resistance against blending.

We evaluated Spectrogram against a range of web-attacks and demonstate strong performance in detecting many real world exploits. Spectrogram achieves a 97% detection rate in unbiased experiments, which uses unique samples, on all but one attack vector (the one that didn't require actual malcode, explained in later sections) and FP rates at five orders of magnitude less when evaluated on the full datasets. This is explained in detail in later sections.

1.1 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 derives the Markov mixture-model which

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 the *active*-defense area of the Network Security and Intrusion Detection Systems (IDS) communities can be broken down into the following broad directions: signature based approaches which attempt to extract ever-present artifacts from malicious code, anomaly detection approaches which detect deviations from accepted input, dynamic analysis which shunts network traffic through an emulator or instrumented host and observe any malicious behavior and tainted data-flow analysis which tries to detect role-violations such as if network traffic reach certain areas of memory which it should not. Hybrids of these approachs have been investigated as well. *Passive*-defensive measures operate at the application and operating systems layer and include measures such as address-space and instruction-set randomization, non-executable stacks, and other memory protection schemes. Spectrogram falls into the former category. In this section we describe some of the notable examples of this category of defense, discuss what the attacker community has developed to counter such methods and explore the area of research that we focus on, which is web-layer (port-80 specific) defense to protect CGI-level applications.

2.1 Signature Based Approaches

Snort [31] is a well known, signature-based detector which operates at the network packet-level (with re-assembly features). It scans through packets to look for strings which are known to be associated with malicious intent. Exploring how to automatically generate exploit signatures has been the focus of a great deal of research. Kim *et. al.* [15] presented one approach to learn such signatures, as did Singh *et. al.* who showed how to extract signatures from worm traffic [30], the Polygraph [24] engine presented by Newsome *et. al.* is another example of solutions which focus on discovering salient artifacts within instances of malware so that signatures may be extracted automatically. This area of research has been well studied [20, 38, 37]. Though these methods all approach the problem from unique perspectives, they share a common theme examining content, or characteristics of network traffic or to add instrumentation to the host to identify malicious input. Reactive host-based learning approaches such as the FLIPS [22] filter traffic through an instrumented version of the application which has some form of a monitoring agent present to detect malcode – if an attack is confirmed, the malcode is dissected to dynamically generate a signature. Cui *et. al.* introduced ShieldGen [8], which uses an instrumented host to dynamically discover vulnerabilities and automatically generate patches. Statistical content anomaly detection is another avenue of research, and PayL [35] models the 1-gram distributions of normal traffic using the Mahalanobis distance as a metric to gauge the normality of incoming packets. Anagram [14] caches known benign n-grams extracted from normal content in a fast hash map and compares ratios of seen and unseen grams to determine normality.

Other approaches include treating all data as potentially carrying executable code and dynamically executing them. Abstract Payload Execution (APE) [33] is a representative example of this line of research. APE examines network traffic and treats packet content as machine instructions. Through the use of dynamic decoding of packet content, APE aims to identify NOP-sleds (blocks of instructions commonly used in shellcode injections). Krugel *et. al.* [19] detect polymorphic worms by learning a control flow graph for the worm binaries with similar techniques. Convergent static analysis [4] also aims at revealing the control flow of a random sequence of bytes. The SigFree [36] system also adopts similar processing techniques. 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.

2.2 Polymorphism and Blending

Recent work [34] calls into question the ultimate utility of exploit-based signatures. Song *et. al.* [32] recently presented a study on the effacy of modern polymorphic techniques and their potential impact on signature and statistical based sensors. Vulnerability-specific protection techniques have be studied [6, 3, 13]. These methods (and especially dynamic taint analysis [5, 25]) explore techinques for defeating exploits despite differences between instances of their encoded form with an alternative detection theory which relies on capturing the characteristics of the vulnerability itself (such as a conjunction of equivalence relations on the set of jump addresses that lead to the exploit: *i.e.,* the control flow path). 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. Crandall *et al.* [6] discuss generating high quality vulnerability signatures via an empirical study of the behavior of polymorphic and metamorphic malcode.

2.3 Web-Layer Defense

The approaches outlined in the preceding subsections are dedicated mainly to defeating memory corruption/code-injection based attacks such as buffer-overflow and heap-corruption. Web-layer attacks, on the other hand, are aimed at CGI-layer data manipulation exploitation and do not require corruptions of the application control-flow (at least not at the memory layer.) On the offensive side, Metasploit has introduced the eVade O'Matic Module (VoMM) [23] which features white-space randomization, string obfuscation/encoding, pseudo-random modification to any comments, block randomization, variables and function name randomization, integer and variables obfuscation and function pointer re-assignment. Van Gundy *et. al.* described a design for a polymorphic PHP worm [12]. MPack [26] is an automated browser exploitation tool that which injects invisible iframes into target servers which in turn silently redirects viewers to exploit-laden websites. These engines inject obfuscation primarily to thwart signature based methods.

On the defensive side, Reis *et. al.* introduced BrowserShield [27] which extends the earlier work of Shield [34] 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, protecting the webserver from malicious users, the previously mentioned PayL [35] and Anagram [14] sensors have been applied to this domain. Spectator [21] was introduced by Livshits *et. al.* as a taint-analysis based approach which tags scripts to prevent Javascript XSS based worms. Kruegel *et. al.* also explored a statistical anomaly detection framework for web traffic [17, 18] based on modeling script inputs, similar to our own approach. We elaborate on the similarities in section 3.

3 Spectrogram

This section contains descriptions for our threat model, the components of the web request to examine, the architectural design of our sensor and the motivations for these choices. We favored

(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=<script language=javascript>alert("Our website is moving!
Please re-login at our new location: www.vulnerable2.com to access the file server!");</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/*

Fig. 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.

an AD approach over modeling malcode (in signature form or otherwise) as a more realistic IDS solution since script writers can provide samples of acceptable traffic (for example, what they used for debugging), whereas attempting to model malcode would force us into an adversarial game where we would be at a disadvantage [23]. Spectrogram, like Snort, is a passive network situated sensor, allowing it to look at remote hosts. However it uses dynamic packet re-assembly to operate at the CGI layer and contains a forensics model to examine Apache log files as well. Web input must conform to acceptable structure as well as content to avoid an alert from this sensor.

3.1 Environment and Threat Model

Web-layer code injection works as follows: a target web-page would contain a script that reads input from the user. Consider, for example, a script that 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 exploit vectors which might succeed if there is no data sanitization (the figure shows `GET` requests since the parameters are visible in the URL). The variable name of the script is "paperID", which takes an integer input value such as "302". If no sanitization is done for the input then an 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." The attacker may also steal a file from the server's disk as shown in Figure (1)(c) in a "local file inclusion" attack. If the server echos the request back in some unsafe way with an error message such as "paperID: xxx not found" where "xxx" is what the input was, then the attacker may execute a cross-site-scripting (`XSS`) attack as shown in Figure (1)(d) to trick other people into visiting the vulnerable site with the malicious code in the URL. This causes the code to appear as if it originated from the victim site. In Figure (1)(e) the same `XSS` goal is accomplished more stealthily with an `iframe` injection to redirect the user to a foreign site which hosts additional browser exploits. Finally, Figure (1)(f) Shows a standard SQL-injection attack which attempts to print the elements of some restricted table. If `POST` is used as the input method then the XSS victim would never even see the injected attack string since it would be in the HTTP message body instead of the URL. Spectrogram monitors all HTTP requests, dynamically re-constructs them from network-layer packets, parses the requests to extract the argument strings and evaluates these strings using pre-trained models for acceptable input.

3.2 Architecture Design

Spectrogram is a passive network-situated IDS solution (NIDS). This choice of positioning is important to allow monitoring of remote hosts, requiring only a port-mirror or deployment at a proxy junction. Unlike other NIDS though, it operates *above* the packet layer, at the CGI-layer, and was designed from the beginning to be HTTP-protocol aware. Dynamically re-assembly allows us to avoid fragmentation attacks and to reconstruct the full HTTP requests as they would be seen by

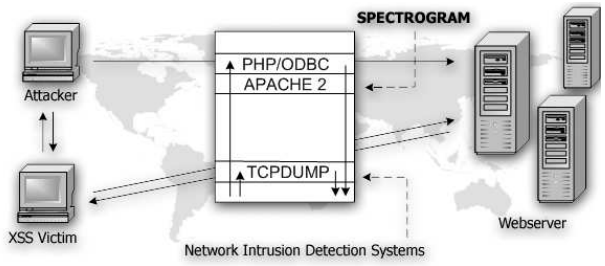


Fig. 2. Spectrogram is a web-layer, network situated sensor.

the targeted web application. The request is then parsed to isolate the argument string and throw away irrelevant content that might skew the sensor (see Section 5 for runtimes).

Without parsing, blending attacks on web-layer AD sensors would be possible by dropping in good content in the unused protocol fields to skew the anomaly score. We have no need for the information in fields such as the `referrer` or `user-agent` strings and since there aren't any Apache or IIS overflows in those fields (that we know of) we ignore these fields and certain others to prevent the anomaly score from being influenced by blending attacks. For further discussion, we refer the reader to Fogla *et. al.* [16]. Spectrogram can examine both `GET` and `POST` requests: for `GET`, we look in the URI to obtain the argument string; for `POST`, the message body. If the web-server allows input in other ways, a new parser can be added. Upon extraction, the argument string is unescaped and put through a normalization function before being passed into the AD module for classification. We elaborate on the latter two of these steps in the next section. Related sensors [35, 14] operate at the network layer and perform packet inspection without protocol-awareness. Thus, to some degree, they can be frustrated by evasion tactics such as packet fragmentation [29] of which `Overlaying re-assembly` is an example. If an attacker knows that the target is running a NIDS on port 80, he may first send a fragmented packet with another port as the destination, a simple firewall or NIDS might ignore this packet. Then, in the second fragment, a marker is set with a low index position. When the upper layers of the network stack re-assembles the packet, the second fragment largely overwrites the first and the overwritten portion would include the fake destination port, replacing it with port 80. In simple implementations, this attack would by-pass NIDS sensors completely.

In addition to network traffic, Spectrogram can also be trained from, as well as do anomaly detection in Apache log files. The completed requests that these logs show lets us avoid the need for dynamic reconstruction. We built our sensor on top of `tcpflow` [11], a packet re-assembly engine for TCP. With each parsed request, the entire argument string is extracted and modeled – both variable names and argument values. The combination of the two elements induces a third important feature: *structure*. This is because with HTTP requests, the two fields are positioned one after the other from left to right. By modeling the entire argument string, we not only learn what the values should be, but also where they should be relatively positioned as well. In essence we want to learn a higher order model for the protocol of the particular script in addition to the content. The next section describes a machine learning based model crafted specifically for such a learning task. The statistical model we use expands upon earlier works by Kruegel *et. al.* [17, 18] whose sensor contains a component which modeled content using single-gram transitions and performed anomaly detection in log files. Our work was designed to be a higher order, more generalized version and used in a *supervised*-learning based, off-host but online sensor which models inputs based on their *overlapping* n-grams. Unsupervised learning based sensors [17, 18, 35, 14] are only guaranteed to find statistical anomalies – inputs not frequently seen but not necessarily malicious. Spectrogram's supervised learning model attempt to fit our sensor tightly to the class of acceptable input.

```
GET /path/script.php?[val1=bleh&val2=blah&val3=...] HTTP/1.1
Host: vulnerable.com
User-Agent: Mozilla/5.0 (Windows; U; Windows NT 5.1; en-US; ...
Accept-Language: en-us,en;q=0.5
Referrer: http://somesite.net
...
```

Fig. 3. Bold, bracketed text shows the portion of the `GET` request we model.

4 The Spectrogram Model

The Spectrogram Model: *For each character of a given input string, the likelihood of that character is conditioned on the previous $n-1$ characters. A single Markov chain calculates the geometric mean of the likelihoods of the individual characters within a string. Spectrogram uses linear mixture of several individual Markov chains with their respective mixing weights as the coefficients. Given any input string, a single likelihood score, which gauges how normal a request seems, is returned. The only parameters to adjust are the gram size, n, and M which controls the number of mixtures. These are set during training.*

Figure (3) shows the portion of the `GET` request which we are interested in modeling – parameter names, valid inputs, as well as their layout with respect to each other are modeled together. For `POST`, the message body is extracted. We would like to know not only that "bleh" is a valid input for "val1" but also that it should follow it in sequence, and in turn be followed by "val2." The Spectrogram model tracks the *n-gram level transitions* of a string – p('al1=bleh&'|'val1=bleh'). Both *content* and *structure* are modeled jointly to not only detect invalid input but also invalid structure, an advantage over previous counting-features based models such as [35, 14] which are agnostic to structure. The following sub-sections illustrates this in more detail. In the next two subsections, we derive a new probability model based on mixtures of higher order Markov-chains, and justify why this is an appropriate choice for our problem of interest. In section 4.4 and the appendix, we derive a learning algorithm which automatically recovers all of the model parameters from the training data.

4.1 N-Grams and the Curse of Dimensionality

N-gram based models have been successfully utilized in recent years in AD roles. An example of such a sensor is `Anagram`, introduced by Wang *et. al.* [14]. Given a string "http://", 2-gram tokens would be "ht", "tt", "tp", *etc.* Such models seek to recover an accurate estimation of the distribution of these grams. *The optimal way to model such a distribution however remains an open question.* Given any string, if we were to assume independence between the individual characters, we can calculate the frequency of each individual character, requiring 256 numbers (byte range). This is what the PayL sensor uses. If we were to model the frequency of n-grams jointly such as the frequency of "http://", we would need 256^7 numbers and in general 256^n numbers 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 – small deviations in the training data can lead to dramatic performance fluctuations in the resulting classifier. 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 power by increasing gram size. Conversely, when modeling the 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 any structure information. As an attacker, one can simply add three "B"'s to the end of the attack string to make the request seem more legitimate.

Spectrogram, on the other hand, 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 $n-1$ preceeding 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, and then "p" two positions back, "t" three positions and so on. Upon examining "val1=BBB&val2=", the "BBB" gram is unrecognized but the "val1=" and "&val2=" gram are. 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. In addition, Spectrogram models string lengths to detect padding. The combination of these efforts resists statistical blending attacks. If an attacker aims to create a blending attack, 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 this 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 requires $256^2 \times (n-1)$ numbers for models at the n-gram level. Since Spectrogram is a mixture of Markov chains with mixing weights, we require $M \times (256^2 \times (n-1)) + M$ numbers 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. In practice, cross-validation can be used to determine the optimal M. The following sub-sections explore the mathematics of the Spectrogram model and the details of our training algorithm.

4.2 Factorized N-Gram Markov Models

An efficient way to model 2-grams is to look at the 1-gram transition likelihoods – the odds that any gram were to follow any other. This is a conditional probability model, and we denote it by $p(x_i|x_{i-1})$, where x_i indicates the i^{th} character of an string and x_{i-1}, the $(i-1)^{th}$ character. This can be generalized to n-grams by conditioning the likelihood of x_n on the $(n-1)$ characters preceding it. More generally, we have $p(x_n|x_{n-1}, x_{n-2}, .., x_1)$. Modeling n-grams without this factorization yields an exponentially growing sample space. For small gram sizes this might be feasible but, in practice, a more elegant solution is required. The Markov chain approach relaxes the distribution to assume *pair-wise independence* – the previous $n-1$ characters are decoupled from *each other* given the n^{th} character ($x_i \perp x_j | x_n$, where $i, j < n$). The final likelihood is set as the *product* of these pair-wise conditional probabilities. For a 5-gram model, this yields $p(x_5|x_4, .., x_1) = p(x_5|x_4)p(x_5|x_3)p(x_5|x_2)p(x_5|x_1)$. Equation (1) gives the likelihood of any character within a string given a model with gram-size G:

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

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

With this factorization, $n-1$ transition matrices (each of dimensionality 256×256) needs to be kept in memory. This yields an algorithm which has complexity growth in $O(n)$. For the likelihood of the whole string, we would like to know the total likelihood of all of the characters thus we need

the product of their respective likelihood values; this is shown in Equation (2) where capital N is used to denote the length of the whole string where we shift the single n-gram window across the string. Since this model is a continuous product of likelihood values (each between 0 and 1), we can see immediately that longer strings will yield lower total likelihoods which is not what we want. What's more appropriate is the *mean* of likelihood values. Since we multiply the likelihoods, we need to find the N^{th} root of the product *i.e.* solve for it's 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 Markov-chain likelihood value for a string. Since we are building models for many classes of scripts, across possibly many hosts, it is insufficient to use only a single chain. To improve the capacity of our model, Spectrogram uses a *mixture* of these Markov chains in order to more accurately represent the many potential classes of input encountered.

4.3 Mixture of Markov Models

To build a mixture model, we use a machine learning procedure of introducing "hidden" states in a weighted-summation mix of the individual chains. Each of these sub-models (p_G) has the form shown in Equation (3). New samples would be evaluated over all M models, then their values combined in a weighted sum to get the ultimate likelihood value. Though they share identical structure, these chains have distinct – and independent – model parameters. We use θ_i to denote the parameters of the i^{th} chain and $\Theta = \{\theta_1, \theta_2, .., \theta_M\}$ to denote a set of parameters for M chains. Each θ_i consists of the $n-1$ transition matrices for that chain with gram-size n. $p(x_i|x_j)$ is the likelihood of a transition from one character to another, and is a single number within one of these matrices. The scalar mixing proportion for a chain indexed by s is denoted by π_s. Summing over these models 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. 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, that is to say, the likelihoods must be normalized.

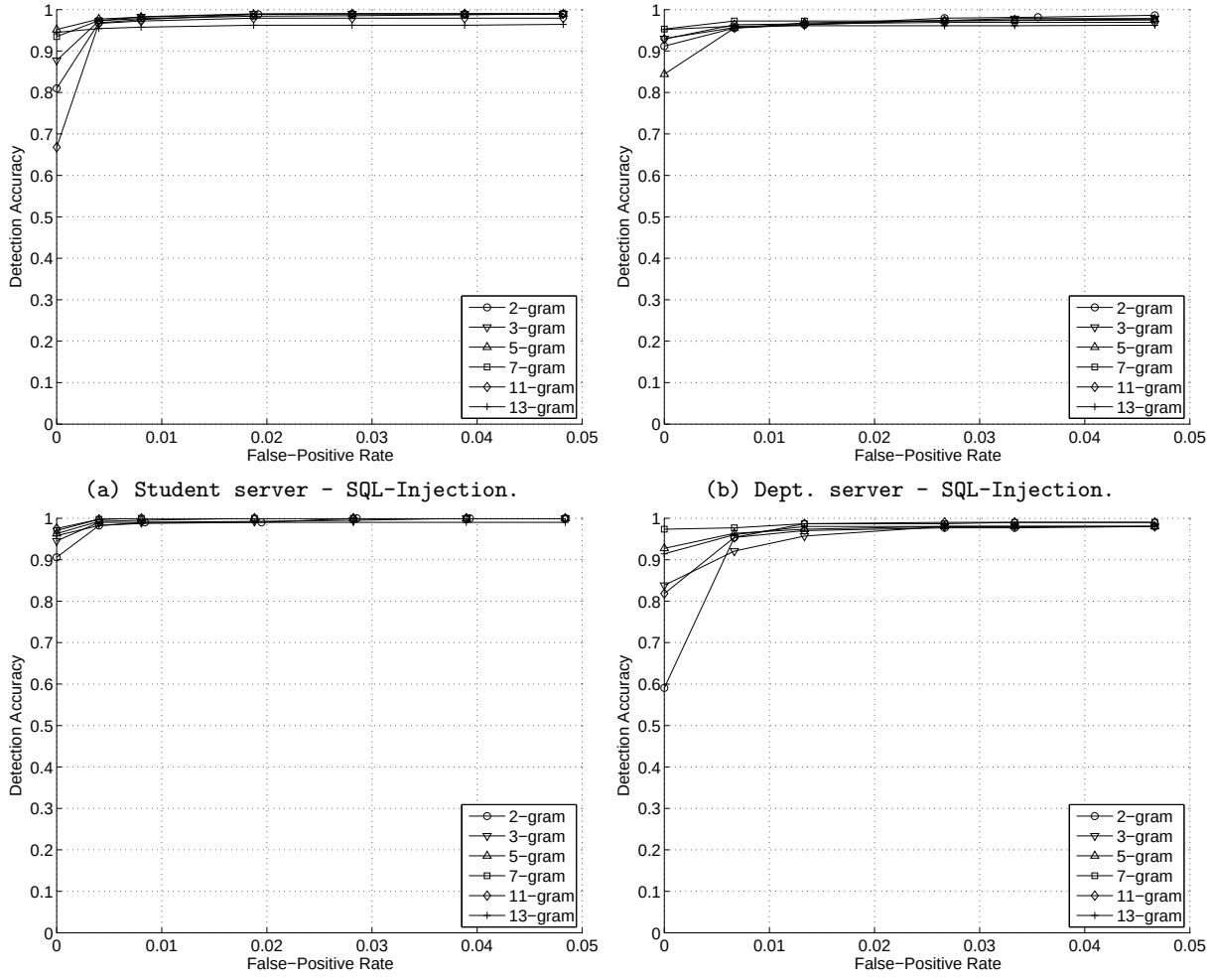
4.4 Training the Mixture Model

For a single Markov-chain, the likelihood function is concave in the data and parameters which means that a single optimal solution exists and can be recovered by setting the first order derivatives to zero and solving. This resolves to counting the number of gram-to-gram transitions within the data, then normalizing the matrices so that each row sums to 1. A mixture of Markov-chains however is *not* concave – 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 complicated. To train a mixture model, we utilize a machine learning procedure known as Expectation Maximization (EM). The mathematical mechanics of this procedure are described in detail in the appendix and we provide pseudo-code for the algorithm. For those not interested in

the details, it suffices to say that it's a gradient ascent algorithm and we iteratively maximize a lower-bound on this function until no improvement is noted in the estimated parameters.

Data Normalization: To improve performance, a reduction in the amount of un-useful features within the data is needed to improve the signal-to-noise ratio. This procedure is as follows: Spectrogram first unescapes each string, then remove all white-space and numbers, then puts the string into lower case. Unescaping makes the models tighter and the training more stable, by making samples from different classes appear as similar to each other as possible. It also serves to to mitigate the efficacy of some obfuscation methods. Stability is the goal, and in deployment the procedure should be tweaked depending on the type of data on the monitored web server.

5 Evaluation



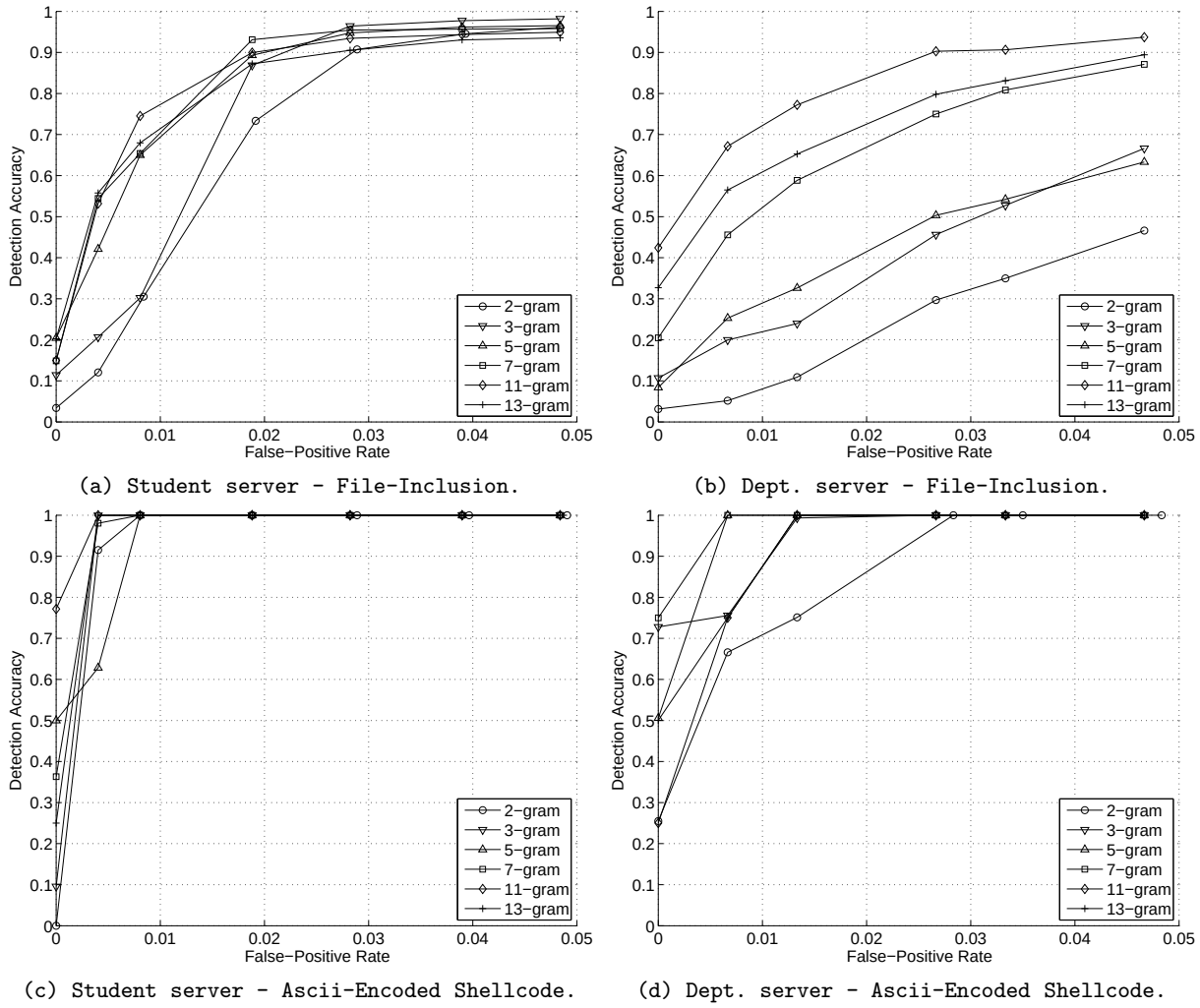
(a) Student server - SQL-Injection.

(b) Dept. server - SQL-Injection.

(c) Student server - Javascript XSS.

(d) Dept. server - Javascript XSS.

Fig. 4. ROC - Spectrogram performance on two unique university servers against different classes of web-based attacks. Observe that increasing gram size improves performance. Earlier works [17, 18] are partially represented by the 2-gram model.



(a) Student server - File-Inclusion.

(b) Dept. server - File-Inclusion.

(c) Student server - Ascii-Encoded Shellcode.

(d) Dept. server - Ascii-Encoded Shellcode.

Fig. 5. ROC - The only poor performance case is on the PHP file inclusion attacks. This is because this particular class of attack doesn't require malcode to be injected but only the *address* of the malcode, making them much harder to detect. Earlier works [17, 18] are partially represented by the 2-gram model.

Scope: We evaluated Spectrogram on two of our university's web-servers. One of the machines hosts our department's 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. A single Spectrogram model is trained per server. Two approaches for evaluation 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 false-positive rate would be biased in our favor, and vise-versa. With **full sampling**, we evaluate Spectrogram using the full set of requests seen, giving us a look at the raw false positive rate when the sensor is deployed for a particular server. All of the attack samples are

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 Shcode	100%	100%	100%	100%
Code-Red	√	√	√	√
Code-Red II	√	√	√	√
IIS-Media	√	√	√	√
IIS-Webdav	√	√	√	√

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 1. Accuracy comparison over the two *unbiased* datasets. Using unique samples lets us gauge model capacity without bias from the data distribution. (S) denotes the student server and (D) denotes the department server. The left tables shows results when we hold the FP rate at 1%. The worms were detected by all of the sensors. In the right table, we show evidence of how Anagram easily underfits when we increase gram-size. The high false positive rates can be expected since the input is short and very dynamic. Spectrogram (`SG-5`) achieves the same level of detection while remaining at 1% FP. See Table 2 for FP rates on the *full* datasets.

unique. **Training**: Spectrogram, as an AD sensor, does not use *any* attack samples in its training, only normal requests. The detections described in this paper are on unseen attack code. When evaluating false-positive rates, we use *unseen* legitimate requests; with unbiased sampling, these legitimate requests are not used during training. In order to avoid false positives, the sensor must infer structure and content-normality by looking at *other* acceptable requests. **Average-of-five**: Every reported number in this paper results from an average of five independent trials. For each trial, the dataset of normal requests is randomly split into 95% training and 5% testing disjoint unique sets. We use all of the attack code for each trial.

5.1 Data

The dataset we used 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 (4.4) 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. 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 [32] polymorphic engines from the Metasploit framework. In addition, we passed these samples through ShellForge's [2] ascii encryption engine to generate 2000 more 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[1]. Each attack is unique.

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 dataset is randomized at this ratio and the average of five trials are reported. The Anagram results highlight the main problem of under-fitting when we increase gram-sizes to increase the power of the classifier. The results show that Anagram is

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

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

Table 2. 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 gauge the capacity of the classifier without bias from the sample distribution.

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 3. Run-times for `SG-5` on a 3Ghz machine. Training is done in MATLAB; 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.

detecting the attacks with higher accuracy when we use larger grams but at the same time it can no longer generalize well and essentially alert on everything. Contrast this result with Spectrogram, whose Markov chain factorization admits much higher gram sizes while maintaining low false-positive 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; for the department server, 2 FPs. Figure (4) and (42) show continued improvement (larger accuracy at the same FP rate) as we increase the gram size. The different sensors were all evaluated in the same manner. We note in passing that Anagram was never designed for this type of string modeling task, but this choice of experiments highlight the problem of n-gram modeling well. Good results were noted in all of the experiments except the `L/RFI` attacks. This is because PHP file inclusion attacks don't require actual attack code, just the *address* of the code, which is then retrieved and executed in the context of the vulnerable process. Because of this, detection is much harder, but still possible as Spectrogram infers some structure on the destination addresses and do recognize most of the injections (also c.f. our discussion on evasion tactics).

Figures (4) and (5) show the ROC curves for `SG-5` when we evaluated it 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.* [17, 18]. The 2-gram curves *partially* represent their sensor which is not publicly available. We say partially because 2-grams is only one part of their framework, at the same time we used 2-grams in a mixture model while they did not. An entirely fair comparison is not possible (especially since their sensor only operates on log files). We include the 2-gram plots to drive the point that *larger gram-sizes improves performance.* Note that while Anagram's FP rate jumped into the 90's with only 3-grams on the unbiased dataset, Spectrogram in most cases showed continued improvements in accuracy 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 3 shows the runtimes 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 didn't fit the data well and thus the training took longer to stabilize. Reconstructing web-layer requests from network packets takes virtually no time when using the `tcpflow` [11] library, which is easily capable of reconstructing nearly 40,000 requests per second on a 3Ghz machine.

5.3 Discussion

Mixture and gram sizes: As we can see from the ROC curves of Figure (4) and (5), there is an obvious benefit to be gained from using larger gram sizes. Clear improvements are observable over the 2-gram model studied earlier [17, 18]. The ability to model 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 seen 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 can be recovered through cross-validation. Tools to automate the cross-validation process will be made available. The threshold setting can be automaticall adjusted based on the false positive rate on the training data.

Clean training data: 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. The optimal way to train the sensor however, is to have the script writers generate samples of legitimate requests, which they would already do when testing their code. Only unique samples of legitimate requests are needed. Furthermore, we refer the reader to the work by Cretu *et. al.* [7] on automated data-sanitisation and labeling for supervised training algorithms.

Re-Training and and Ranking: Spectrogram's model doesn't admit online/continuous training. However, as we've shown, it takes only a few of minutes to train this model using over 15,000 samples. Automated nightly or even hourly re-training is not unreasonable to deal with script updating – all that's required is for an admin to notice false positives in the logs and to extract and insert these into the training set. This methodology has more flexibility than, say, relying on Snort signatures. We can also isolate hosts based on their update frequency and schedule their re-training appropriately if resources are limited. Spectrogram outputs a normality score for each request and it's possible to rank alerts to generate short-lists for human analysis.

Parallelization and Forensics: Spectrogram is a standalone AD sensor and, like Snort, it is possible to deploy several sensors on the network in parallel to monitor different subnets or individual hosts, as would be the case when protecting large data-centers. The design choice of a passive, remote sensor adds to this flexibility. The small memory footprint allows multiple instances to be run on the same entity if needed, for example if the deployment was on a gateway with more than one network interface. Parallelization of a particular sensor is as easy as swapping the trained model-files between sensors. An offline-mode for anomaly detection in Apache log-files is also available for forensics.

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 evaded by fragmentation attacks; Spectrogram dynamically re-assembles requests to reconstruct the full attack – to see what the target script would see. Polymorphism can fool signature based sensors; Spectrogram utilizes anomaly detection

and never trains using malicious content, therefore polymorphism has no effect as we can see in Table (1). Counting-features based statistical AD sensors can have their scoring skewed by attacks crafted to appear like normal requests; Spectrogram models high-order *structure* as well as content, 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 and not an attack. Unfortunately, our sensor is not completely fool-proof: if the protected scripts legitimately reads data from foreign sources with only destination addresses exposed, data-sanitization must exist within the scripts themselves. Spectrogram is a passive sensor and will not perform data validation beyond CGI-layer inputs. Therefore it can't tell if the foreign site will attempt to exploit the script or not. This is the main reason why PHP file inclusion attacks are much harder to detect in our experiments since they required only sending the addresses of the exploits and not the actual exploits themselves.

6 Conclusions

In conclusion, Spectrogram is designed to protect web-servers at the CGI-layer, as well as memory level shellcode injections which target the web-server using HTTP requests. Like Snort, it is a passive sensor, designed with remote monitoring in mind, except it was designed from an anomaly detection approach. This paper discussed why N-gram modeling is exponentially hard in its naive form and demonstrated this phenomenon in our experiments. We derived a relaxation of n-gram modeling into a more tractable linear solution with Markov-chains and introduced a new probability model for n-gram-level transitions within argument strings. We further derived the parameter estimation techniques to train this model. Spectrogram has two adjustable parameters: the mixture-size and the gram-size; the optimal settings for both can be found by cross-validation to obtain the desired trade-off between speed and accuracy, depending on the content on the server. Our empirical results demonstrate the efficacy of using this method in a web-layer anomaly detection framework by achieving 97% detection accuracy on all but one of the unbiased datasets at 1% false-positive rate, comparing favorably with related sensors. Our runtime results show that this sensor is fast enough for use in online environments and flexible enough for multiple roles and deployment postures.

7 APPENDIX

Training the Spectrogram model: Let $p(\mathcal{D}|\theta)$ denote the likelihood of observing a dataset of training samples, represented as $\mathcal{D}$. We know from Bayes' Theorem that the optimal parameter, θ, is the one that maximizes this function. If $p(\mathcal{D}|\theta)$ is concave in parameter space then there is a unique optimal solution and we can recover it by solving the gradient of $p(\mathcal{D}|\theta)$ with respect to θ, setting the gradient to zero and solving for θ. For a single Markov-chain, the function is in-fact concave, and the solution for the optimal θ entails counting the number of *gram-to-gram* transitions within the training data then normalize the matrices so that each row sums to 1. $P(\mathcal{D}|\theta)$ for mixture of Markov-chains however, is *not* concave – the hidden states remove the concavity because summations over concave functions do not preserve concavity. This removes the unique-optimal-solution property. To train this mixture model, we must instead make use of an alternative procedure known as Expectation Maximization (EM). EM is a popular parameter estimation technique in machine learning. First introduced by Dempster *et. al.* [9] in 1977, it is a procedure for optimizing non-concave functions through gradient ascent. It contains two core steps: the `Expectation` step (E-step) calculates the likelihood of observing the training set given

the current set of model parameters Θ and the `Maximization` step (M-step) finds the gradient of a concave lower-bound on the likelihood function and offsets the parameters in the direction of the gradient, thus maximizing $P(\mathcal{D}|\theta)$ iteratively at each step. These two steps guarantee monotonic convergence to a local-max and can bee alternated until no gain is noticed. The EM update rules must be derives on a per-model basis; we derives the rules for our model below:

E-STEP: In the E-step, we need to solve $P(\mathcal{D}|\theta)$, 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 $\mathbf{x}_d$ indicates an arbitrary lengthed string in $\mathcal{D}$. Next we need a lower bound on the expected value. For this we invoke `Jensen's inequality` which states that given a concave function $f(x)$, we have $f(\sum x) \geq \sum f(x)$. We use log for $f(x)$ and instead of solving for Equation (5) directly, we solve $\log\left(\prod_{d=1}^{|\mathcal{D}|} p_G(\mathbf{x}_d|\Theta)\right)$. Since logarithms are monotonic transformations, the optimal parameter are equivalent for both functions:

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

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 required lower bound.

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

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

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

Equation (8) is our lower bound. 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 the model parameters Θ and the mixing weights $\{\pi_1,..,\pi_M\}$. Given the previously mentioned constraints on the transition matrix, specifically 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 will be made available at a later time on our website.[2]

The M-STEP proceeds as follows: let $\tau_{d,s}$ denote the log-likelihood of observing string $\mathbf{x}_d$ given model θ_s. 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

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

chains.

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

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

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

The entire training algorithm for Spectrogram's statistical model is to alternate between these two E and M steps. Figure (6) shows the pseudo-code for the training process. The algorithm accepts

```
function train-spectrogram (D, G, M)
1   {π_1, π_2, .., π_M}, Θ = {θ_1, .., θ_M} ← randomly-initialize
2   Z_1 ← Equation (8)
3   for i = 2 to ITER-LIMIT
4      Update {π_1, .., π_M} using Equation (9)
5      Update Θ using Equation (10)
6      Z_i ← Equation (8)
7      if(Z_i − Z_{i−1}) < T then break
8   done
return {π_1, π_2, .., π_M, Θ}
```

Fig. 6. 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.

as input, the training dataset $\mathcal{D}$, the gram-size G and the number of Markov-chains M to use. The output is the full mixture of Markov chains model, ready to score new requests.

References

1. Anagnostakis, K. G., Sidiroglou, S., Akritidis, P., Xinidis, K., Markatos, E., and Keromytis", A. D. Detecting Targeted Attacks Using Shadow Honeypots. In *"Proceedings of the 14^{th} USENIX Security Symposium."*.
2. Biondi", P. Shellforge Project, "2006".
3. Brumley, D., Newsome, J., Song, D., Wang, H., and Jha", S. Towards Automatic Generation of Vulnerability-Based Signatures. In *"Proceedings of the IEEE Symposium on Security and Privacy"*.
4. Chinchani, R., and Berg", E. V. D. A Fast Static Analysis Approach to Detect Exploit Code Inside Network Flows. In *"Proceedings of the 8^{th} International Symposium on Recent Advances in Intrusion Detection (RAID)"*, pp. 284–304.
5. Costa, M., Crowcroft, J., Castro, M., and Rowstron", A. Vigilante: End-to-End Containment of Internet Worms. In *"Proceedings of the Symposium on Systems and Operating Systems Principles (SOSP)"*.
6. Crandall, J. R., Su, Z., Wu, S. F., and Chong", F. T. 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)"*.
7. Cretu, G. F., Stavrou, A., Locasto, M. E., Stolfo, S. J., and Keromytis, A. D. Casting out demons: Sanitizing training data for anomaly detection. In *Proceedings of the IEEE Symposium on Security and Privacy (S&P)* (May 2008).
8. Cui, W., Peinado, M., Wang, H. J., and Locasto", M. E. ShieldGen: Automated Data Patch Generation for Unknown Vulnerabilities with Informed Probing. In *"Proceedings of the IEEE Symposium on Security and Privacy"*.

9. Dempster, A., Laird, N., and Rubin, D. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society* (1977), 1–38.
10. Dubrawsky, I. Data Driven Attacks Using HTTP Tunneling, 2004.
11. Elson, J. tcpflow – A TCP Flow Recorder, 2003. http://www.circlemud.org/ jelson/software/tcpflow/.
12. Gundy, M. V., Balzarotti, D., and Vigna, G. 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).
13. Joshi, A., King, S. T., Dunlap, G. W., and Chen", P. M. Detecting Past and Present Intrusions through Vulnerability-Specific Predicates. In *"Proceedings of the Symposium on Systems and Operating Systems Principles (SOSP)"*.
14. Ke Wang, Janak J. Parekh, S. J. S. Anagram: A Content Anomaly Detector Resistant To Mimicry Attack. In *"Proceedings of the* 9^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)"*.
15. Kim, H.-A., and Karp", B. Autograph: Toward Automated, Distributed Worm Signature Detection. In *"Proceedings of the USENIX Security Conference"*.
16. Kolesnikov, O., and Lee, W. Advanced polymorphic worms: Evading ids by blending in with normal traffic. In *Proceedings of the USENIX Security Symposium* (2006).
17. Kruegel, C., and Vigna, G. Anomaly Detection of Web-Based Attacks. In *Proceedings of the* 10^{th} *ACM Conference on Computer and Communications Security (CCS)*.
18. Kruegel, C., Vigna, G., and Robertson, W. A Mult-Model Approach to the Detection of Web-based Attacks. *computer Networks "48"* (2005).
19. Krugel, C., Kirda, E., Mutz, D., Robertson, W., and Vigna", G. Polymorphic Worm Detection Using Structural Information of Executables. In *"Proceedings of the* 8^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)"*, pp. 207–226.
20. Liang, Z., and Sekar", R. 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)"*.
21. Livshits, B., and Cui, W. Spectator: Detection and containment of javascript worms. In *"Proceedings of the USENIX Annual Technical Conference"* (June 2008).
22. Locasto, M. E., Wang, K., Keromytis, A. D., and Stolfo", S. J. FLIPS: Hybrid Adaptive Intrusion Prevention. In *"Proceedings of the* 8^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)"*, pp. 82–101.
23. Metasploit Developement Team. Metasploit Project, "2006".
24. Newsome, J., Karp, B., and Song", D. Polygraph: Automatically Generating Signatures for Polymorphic Worms. In *"Proceedings of the IEEE Symposium on Security and Privacy"*.
25. Newsome, J., and Song, D. 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).
26. Panda Labs. MPack Uncovered, "2007".
27. Reis, C., Dunagan, J., Wang, H. J., Dubrovsky, O., and Esmeir, S. Browsershield: Vulnerability-driven filtering of dynamic html. In *Proceedings of Symposium on Operating Systems Design and Implementation (OSDI)* (2006).
28. SANS. SANS Top 20. http://www.sans.org/top20/.
29. Siddharth, S. Evading NIDS, revisited, 2005. http://www.securityfocus.com/infocus/1852.
30. Singh, S., Estan, C., Varghese, G., and Savage", S. Automated Worm Fingerprinting. In *"Proceedings of Symposium on Operating Systems Design and Implementation (OSDI)"*.
31. Snort Development Team. Snort Project.
32. Song, Y., Locasto, M. E., Stavrou, A., Keromytis, A. D., and Stolfo, S. J. On the Infeasibility of Modeling Polymorphic Shellcode. In *"Proceedings of the ACM Conference on Computer and Communications Security (CCS)"*.
33. Toth, T., and Kruegel", C. Accurate Buffer Overflow Detection via Abstract Payload Execution. In *"Proceedings of the* 5^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)"*, pp. 274–291.
34. Wang, H. J., Guo, C., Simon, D. R., and Zugenmaier, A. Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits. In *Proceedings of the ACM SIGCOMM Conference* (August 2004), pp. 193–204.
35. Wang, K., Cretu, G., and Stolfo", S. J. Anomalous Payload-based Worm Detection and Signature Generation. In *"Proceedings of the* 8^{th} *International Symposium on Recent Advances in Intrusion Detection (RAID)"*, pp. 227–246.
36. Wang, X., Pan, C.-C., Liu, P., and Zhu", S. SigFree: A Signature-free Buffer Overflow Attack Blocker. In *"Proceedings of the* 15^{th} *USENIX Security Symposium"*, pp. 225–240.

37. Xu, J., Ning, P., Kil, C., Zhai, Y., and Bookholt", C. Automatic Diagnosis and Response to Memory Corruption Vulnerabilities. In *"Proceedings of the* 12^{th} *ACM Conference on Computer and Communications Security (CCS)"*.
38. Yegneswaran, V., Giffin, J. T., Barford, P., and Jha", S. An Architecture for Generating Semantics-Aware Signatures. In *"Proceedings of the* 14^{th} *USENIX Security Symposium"*.