Anagram: A Content Anomaly Detector Resistant to Mimicry Attack[1]

Ke Wang Janak J. Parekh Salvatore J. Stolfo

Computer Science Department, Columbia University
500 West 120th Street, New York, NY, 10027
{kewang, janak, sal}@cs.columbia.edu

Abstract. In this paper, we present *Anagram*, a content anomaly detector that models *a mixture of high-order n-grams* (n > 1) designed to detect anomalous and "suspicious" network packet payloads. By using higher-order n-grams, Anagram can detect significant anomalous byte sequences and generate robust signatures of validated malicious packet content. The Anagram content models are implemented using highly efficient Bloom filters, reducing space requirements and enabling privacy-preserving cross-site correlation. The sensor models the distinct content flow of a network or host using a semi-supervised training regimen. Previously known exploits, extracted from the signatures of an IDS, are likewise modeled in a Bloom filter and are used during training as well as detection time. We demonstrate that Anagram can identify anomalous traffic with high accuracy and low false positive rates. Anagram's high-order n-gram analysis technique is also resilient against simple mimicry attacks that blend exploits with "normal" appearing byte padding, such as the blended polymorphic attack recently demonstrated in [1]. We discuss *randomized n-gram models*, which further raises the bar and makes it more difficult for attackers to build precise packet structures to evade Anagram even if they know the distribution of the local site content flow. Finally, Anagram's speed and high detection rate makes it valuable not only as a standalone sensor, but also as a network anomaly flow classifier in an instrumented fault-tolerant host-based environment; this enables significant cost amortization and the possibility of a "symbiotic" feedback loop that can improve accuracy and reduce false positive rates over time.

1 Introduction

The current generation of Network Intrusion Detection Systems (NIDS) are typically ill-suited for stealthy worms and targeted attacks. Misuse and anomaly detectors that analyze packet headers and traffic flow statistics may be too slow to react to reliably detect worms that are designed to evade detection by shaping their behavior to look like legitimate traffic patterns [2]. Furthermore, signature scanners are vulnerable to zero-day exploits [3] and polymorphic worms/stealthy attacks with obfuscated exploit code [4]. Consequently, there has been an increasing focus on payload analysis to detect the early onset of a worm or targeted attack. Ideally, one would hope to detect the very first packets of an attack, rather than accumulating sufficient statistics about connection flows to detect a zero-day attack.

A number of researchers (e.g., [5-8]) have focused on payload-based anomaly detection. Approaches that have been studied include specification-based anomaly detection [7] as well as techniques that aim to detect "code-like" byte sequences in network payloads [6, 9]. In our work, we have focused on automated statistical learning approaches to efficiently train content models on a site's "normal" traffic flow without requiring

[1] This work has been partially supported by a grant with the Army Research Office, No. DA W911NF-04-1-0442.

significant semantic analysis. Ideally, we seek to design a sensor that automatically learns the characteristics of "normal" attack-free data for any application, service, network or host. Consequently, a model learned for "normal" attack-free data may be used to identify "abnormal" or suspicious traffic that would be subjected to further analysis to validate whether the data embodies a new attack.

In our previous work we proposed PAYL (short for "PAYLoad anomaly detection") that modeled the "normal" attack-free traffic of a network site as 1-gram, byte-value frequency distributions [10], and demonstrated an ability to effectively detect worm behavior via ingress/egress and cross-site correlation [11]. The sensor was designed to be language-independent, requiring no syntactic analysis of the byte stream. Furthermore, PAYL was designed to be efficient and scalable for high-speed networks and applicable to any network service. Various experiments demonstrated that PAYL achieved a high detection rate and with low false positives for "typical" worms and exploits available at the time.

However, most researchers[2] correctly suspected that PAYL's simplicity would be easily blinded by *mimicry attacks*. Kolesnikov, Dagon and Lee [1] demonstrated a new blended, polymorphic worm designed to evade detection by PAYL and other frequency distribution-based anomaly detectors. This demonstration represents a new class of "smart worms" that launch their attack by first sniffing traffic and shaping the datagram to the statistics specific to a given site to appear normal. The same principles may be applied to the propagation strategy as well as in, for example, parasitic worms. Since PAYL only models 1-gram distributions, it can be easily evaded with proper padding to avoid detection of anomalous *byte sequences*. As a countermeasure, we conjecture that higher-order n-gram modeling may likely detect these anomalous byte sequences. Unfortunately, computing a full frequency distribution for higher order n-grams is computationally and memory-wise infeasible, and would require a prohibitively long training period even for modest gram sizes.

In this paper we present a new sensor, Anagram, which introduces the use of Bloom filters and a binary-based detection model. Anagram does not compute frequency distributions of normal content flows; instead, it trains its model by storing all of the distinct n-grams observed during training in a Bloom filter without counting the occurrences of these n-grams. Anagram also stores n-grams extracted from known malicious packets in a second *bad content Bloom filter*, acquired by extracting n-grams from openly available worm detection rules, such as the latest Snort rulesets [12]. At detection time, packets are scored by the sensor on the basis of the number of unobserved n-grams the packet contains. The score is weighted by the number of malicious n-grams it contains as well. In this paper, we demonstrate that this semi-supervised strategy attains remarkably high detection and low false positive rates, in some cases 100% detection with less than 0.006% false positive rate (per packet).

The use of Bloom filters makes Anagram memory and computationally efficient and allows for the modeling of a *mixture of different sizes of n-grams* extracted from packet payloads, i.e. an Anagram model need not contain samples of a fixed size gram. This strategy is demonstrated to exceed PAYL in both detection and false positives rates. Furthermore, Anagram's modeling technique is easier to train, and allows for the estimation of when the sensor has been trained enough for deployment. The Bloom filter model representation also provides the added benefit of preserving the privacy of shared content models and alerts for cross-site correlation.

[2] Including ourselves; a proposal to study counter-evasion techniques led to the work reported herein.

Of particular interest here is that Anagram is shown to be robust against existing mimicry attack approaches. We demonstrate Anagram's ability to counter the simple mimicry attacks levied against PAYL. Furthermore, Anagram is designed to defeat training and mimicry attacks by using *randomized n-gram modeling*. The approach presented raises the bar against the enemy, making it far more difficult to design an n-gram attack against Anagram. By randomizing the portion of packets that Anagram extracts and models, mimicry attackers cannot easily guess how and where to pad malicious content to avoid detection. We also describe the use of a feedback loop between the Anagram sensor and host-based detectors, thereby updating Anagram models over time and improving its detection performance. Thus, the combination of model randomization and a feedback loop makes it more difficult to evade detection by training and mimicry attacks. The contributions of this paper include:

- A new statistical, language-independent, efficient content-based anomaly detector based upon semi-supervised training of higher-order n-gram analysis that is shown to be resistant against existing mimicry attacks. The sensor does not rely upon a specification or semantic analysis of the target applications;
- Robustness against future mimicry attacks by the use of a novel, low-overhead randomized testing strategy, making it difficult for the attacker to guess *where* or *how* to pad content;
- Development of a run-time measure of the "stability" of a network's content flow, providing a reasonable estimate of when the sensor has been well enough trained and is ready for deployment.
- A robust means of representing content-based alerts for cross-site alert sharing and robust signature generation using a Bloom Filter representation of anomalous byte sequences[3];
- A new defensive strategy showing how a symbiotic relationship between host-based sensors and a content-based sensor can adapt over time to improve accuracy of modeling a site's content flow.

The rest of the paper is organized as follows. Section 2 details the Anagram sensor and its relevant algorithms. Performance, detection rate, and false positive characteristics are presented testing Anagram against real network traffic traces infected with a collection of worms and viruses. Section 3 describes Anagram's robustness against the cleverly crafted blended polymorphic worm reported in [1], previews the possibility of new customized mimicry attacks being crafted against Anagram, and describes randomization techniques for defeating such attacks. In section 4 we present the concept of coupling a "shadow server" with Anagram and discuss how the combination can effectively support the techniques presented in section 3, as well as support robust signature generation and patch generation. Section 5 discusses related work, while section 6 concludes the paper with a call for collaboration among researchers at distributed sites.

2 Anagram – Modeling a Mixture of N-grams

Anagram's approach to network payload anomaly detection uses a mixture of higher order n-grams (n>1) to model and test network traffic content. N-gram analysis is a well-known technique has been used in a variety of tasks, such as system call monitoring [15-17]. In Anagram, the n-grams are generated by sliding windows of arbitrary lengths over a stream of bytes, which can be per network packet, per request session, or other type of data unit.

[3] The representation also permits patch generation systems to share anomalous data for local patch generation across an "application community" [13, 14].

In our previous work on network payload anomaly detection, PAYL [10, 11], we modeled the length-conditioned 1-gram frequency distribution of packet payloads, and tested new packets by computing the Mahalanobis distance of the test packet against the model. This approach is effective at capturing attacks that display abnormal byte distributions, but it is likely to miss well-crafted attacks that focus on simple CPU instructions and that are crafted to resemble normal byte distributions. For instance, although a standard CodeRed II's buffer overflow exploit uses a large number of "N" or "X" characters and so appears as a peak in the frequency distribution, [18] shows that the buffer can instead be padded with nearly any random byte sequence without affecting the attack vector. Another example is the following simple phpBB forum attack:

```
GET /modules/Forums/admin/admin_styles.php?phpbb_root_path=http
://81.174.26.111/cmd.gif?&cmd=cd%20/tmp;wget%20216.15.209.4/cri
man;chmod%20744%20criman;./criman;echo%20YYY;echo|..HTTP/1.1.Ho
st:.128.59.16.26.User-Agent:.Mozilla/4.0.(compatible;.MSIE.6.0;
.Windows.NT.5.1;)..
```

In such situations, the abnormal byte distribution model is insufficient by itself to identify these attack vectors as abnormal data. However, invariants remain in the packet payloads: the exploit code, the sequence of commands, or the special URL that should not appear in the normal content flow to the target application. By modeling higher order n-grams, Anagram captures the order dependence of byte sequences in the network payload, enabling it to capture more subtle attacks. The core hypothesis is that any new, zero-day exploit will contain a portion of data that has never before been delivered to the application. These subsequences of new, distinct byte values will manifest as anomalous" n-grams that Anagram is designed to efficiently and rapidly detect.[4]

In the following sections we will give a detailed description of Anagram, which outperforms PAYL in the following respects:

- Accuracy in detecting anomalous payloads, even carefully crafted 'mimicry attacks' with a demonstrably lower false positive rate;
- Computational efficiency in detection by the use of fast (and incremental, linear-time) hashing in its Bloom filter implementation;
- Model space efficiency since PAYL's multiple-centroid modeling is no longer necessary, and Bloom filters are compact;
- Fast correlation of multiple alerts while preserving privacy as collaborating sites exchange Bloom filter representations of common anomalous payloads;
- The generation of robust signatures via cross-site correlation for early warning and detection of new zero day attacks.

In the following sections, we will describe the mechanisms in detail and present experimental results of testing Anagram against network traces sniffed from our local LAN.

2.1 High Order N-gram Payload Model

While higher order n-grams contain more information about payloads, the feature space grows exponentially as *n* increases. Comparing an n-gram frequency distribution against a model is infeasible since the training data is simply too sparse; the length of a packet is too small compared to the total feature space size of a higher-order n-gram.

[4] Note that some attacks may not include byte sequences that are "code-like", and hence testing content for such code-like data subsequences is not guaranteed to cover all attack cases. The language independence of anomalous n-grams may be broadly applicable to these and other attacks.

One TCP packet may contain only a thousand or so n-grams, while the feature space size is 256^n. Clearly, with increasing *n*, generating sufficient frequency statistics to estimate the true empirical distribution accurately is simply not possible in a reasonable amount of time.

In Anagram, we therefore do not model the frequency distribution of each n-gram. Rather, we observe each distinct n-gram seen in the training data and record each in a space efficient Bloom filter. Once the training phase terminates, each packet is scored by measuring the number of n-grams it did not observe in the training phase. Hence, a packet is scored by the following formula, where N_{new} is the number of new n-grams not seen before and T is the total number of n-grams in the packet:

$$Score = \frac{N_{new}}{T} \in [0,1]$$

At first glance, the frequency-based n-gram distribution may contain more information about packet content; one might suspect it would model data more accurately and perform better at detecting anomalous data, but since the training data is sparse, this alternative “binary-based model” performs significantly better than the frequency-based approach given the same amount of training data.

We analyzed the network traffic for the Columbia Computer Science website and, as expected, a small portion of the n-grams appear frequently while there is a long “tail” of n-grams that appear very infrequently. This can be seen in Table 1, which displays the percentage of the n-grams by their frequency counts for 90 hours of CS web traffic. Since a significant number of n-grams have a small frequency count, and the number of n-grams in a packet is very small relative to the whole feature space, the frequency-distribution model incurs relatively high false positives. Thus, the binary-based model (simply recording the distinct n-grams seen during training) provides a reasonable estimate of how “normal” a packet may be. This is a rather surprising observation; as we will demonstrate, it works very well in practice. The conjecture is that true attacks will be delivered in packets that contain many more n-grams not observed in training than “normal” packets used to train the model. After all, a true zero-day attack must deliver data to a server application that has never been processed by that application before. Hence, the data exercising the vulnerability is very likely to be an n-gram of some size never before observed. By modeling a mixture of n-grams, we increase the likelihood of observing these anomalous grams.

To validate this conjecture, we compare the ROC curves of the frequency-based approach and the binary-based approach for the same datasets (representing equivalent training times) as displayed in figure 1. We collected the web traffic of two CS departmental web servers, *www* and *www1*; the former serves the department webpage, while the latter serves personal web pages. Traffic was collected for two different time periods: a period of sniffed traffic from the year 2004 and another dataset sniffed in 2006. The 2004 datasets[5] (*www-04* and *www1-04*) contain 160 hours of traffic; the 2006 datasets (*www-06* and *www1-06*) contain about 560 hours. We tested for the detection of several real worms and viruses: CodeRed, CodeRed II, WebDAV, Mirela, a php forum attack, and a worm that exploits the IIS Windows media service, the nsiislog.dll buffer overflow vulnerability (MS03-022). These worm samples were collected from real traffic as they appeared in the wild, from both our own dataset and from a third-party.

[5] The 2004 dataset was used to train and test PAYL as reported in [11], and is used here for comparative evaluation.

For the first experiment, we used 90 hours of www1-06 for training and 72 hours for testing. (Similar experiments on the other datasets display similar results, and we skip them here for brevity.) To make it easier for the reader to see the plots, the curves are plotted for the cases where the false positive rate is less than 0.03%. Both the detection rate and false positive rate are calculated based on packets with payloads; non-payload (e.g., control) packets were ignored. Notice that the detection rate in figure 1 is based on per packet, instead of per attack. Some attacks have multiple packets; while fragmentation can result in a few packets appearing normal, we can still guarantee reliable attack detection over the entire set of packets. For example, for the IIS5 WebDAV attack, 5-grams detect 24 out of 25 packets as being anomalous. The only missed packet is the first packet, which contains the buffer overflow string "SEARCH /AAA……AA"; this is not the key exploit part of the attack. For further comparison, we list minimum false positive rates when detecting all attack attempts (where an attack is detected if at least 80% of the packets are classified as anomalous) for both binary-based and frequency-based models in table 2.

Table 1. The percentage of the observed unique n-grams for different frequencies of occurrence for 90 hours of training traffic

frequency count	3-grams	5-grams	7-grams
>=5	58.05%	39.13%	32.53%
2 to 4	22.17%	28.22%	28.48%
1	19.78%	32.65%	38.99%

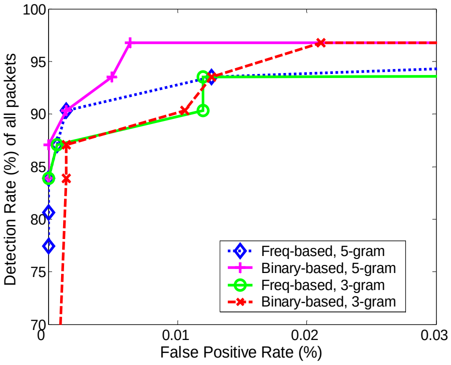

Fig. 1. ROC curves comparing the frequency-based and binary-based n-gram approach

Table 2. The false positive rate (%) of the two approaches using different n-grams when achieving 100% detection rate, www1-06 train/test dataset

	3-gram	4-gram	5-gram	6-gram	7-gram	8-gram
Freq-based	30.40205	0.18559	0.08858	0.13427	0.30792	0.41477
Binary-based	0.02109	0.01406	0.00634	0.00703	0.00914	0.00914

The binary-based approach yields significantly better results than the frequency-based approach. When a 100% detection rate is achieved for the packet traces analyzed, the false positive rate of the binary-based approach is at least *one order of magnitude less* than the frequency-based approach. The relatively high false positive rate of the frequency-based approach suggests much more training is needed to capture accurate statistical information to be competitive. In addition, the extremely high false positive rate of the 3-gram frequency-based approach is due to the fact that the 3-grams of the php attack all appear frequently enough to make it hard to distinguish them from normal content packets. On the other hand, the binary-based approach used in Anagram results in far better performance. The 0.01% false positives average to about 1 alert per hour for *www1* and about 0.6 alerts per hour for *www*. The result also shows that 5-grams and 6-grams give the best result, and we've found this to be true for others as well.

As previously stated, Anagram may easily model a mixture of different n-grams simply by storing these in the same Bloom filter. However, for larger n-grams additional training may be required; as we shall describe shortly, our modeling approach allows us to estimate when the sensor has been sufficiently trained.

2.2 Model Size and Bloom Filters

As previously stated, one significant issue when modeling with higher order n-grams is memory overhead. By leveraging the binary-based approach, we can use more memory-efficient set-based data structures to represent the set of observed n-grams. In particular, the *Bloom filter* (BF) [19] is a convenient tool to represent the binary model. Instead of using *n* bytes to represent the n-gram, or even 4 bytes for a 32-bit hash of the n-gram, the Bloom filter can represent a set entry with just a few bits, reducing memory requirements by an order of magnitude or more.

A Bloom filter is essentially a bit array of *m* bits, where any individual bit *i* is set if the hash of an input value, `mod` *m*, is *i*. As with a hash table, a Bloom filter acts as a convenient one-way data structure that can contain many items, but generally is orders-of-magnitude smaller. Operations on a Bloom filter are $O(1)$, keeping computational overhead low. A Bloom filter contains no false negatives, but may contain false positives if collisions occur; the false positive rate can be optimized by changing the size of the bit array and by using multiple hash functions (and requiring all of them to be set for an item to be verified as present in the Bloom filter; in the rare case where one hash function collides between two elements, it's highly unlikely a second or a third would also simultaneously collide). By using universal hash functions [20], we can minimize the probability of multiple collisions for n-grams in one packet (assuming each n-gram is statistically independent); the Bloom filter is therefore safe to use and does not negatively affect detection accuracy.

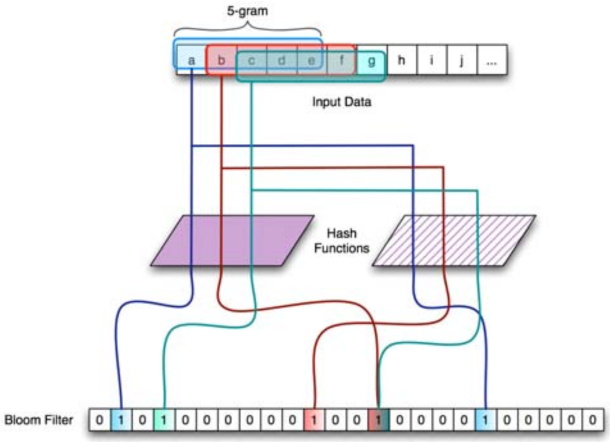

Fig. 2. Inserting n-grams (here, n = 5) into a Bloom filter.

2.2.1 Memory overhead

While Bloom filters are comparatively small even when inserting a large number of entries, choosing the optimal size of a Bloom filter is nontrivial, since Anagram is not aware of a site's distribution (and the number of unique n-grams) before building its model. Additionally, a Bloom filter's size cannot be dynamically resized, as the hash values cannot be recomputed without the original underlying training data.[6]

A large Bloom filter will waste memory, but small Bloom filters saturate more quickly, yielding higher false positive rates. It is worth pointing out that "large" is relative; a 24-bit Bloom filter is capable of holding $2^{24}/n_h$ elements, where n_h represents the number of hash functions used, in only 2MB of memory, e.g., each n-gram inserted uses about 2.7 bits when 3 hash functions are used. Additionally, we can use traditional compression methods (e.g., LZW) for storing a sparse Bloom filter, which significantly reduces storage and transport costs. As discussed later in this paper, our experiments anecdotally suggest this Bloom filter is large enough for at least 5-grams, assuming a mostly textual distribution.

The presence of binary data does significantly increase Bloom filter requirements; if allocating extra initial memory is undesirable, a "layered" approach can be employed, where new Bloom filters are created on demand as previous Bloom filters saturate, with a small constant-time overhead. It should be evident that Bloom filters can be trivially merged via bitwise ORing and compared via bitwise ANDing.

2.2.2 Computation overhead

The overhead of *insert*ing or *verify*ing a single item into a Bloom filter is O(1) per item inserted in the Bloom filter. However, the constant-time overhead to process an n-gram can become significant when inserting or checking n-grams over a large population of packets. This effect is magnified when an Anagram model uses different-size n-grams, as the same data is repeatedly hashed into different positions on the Bloom filter based on the various n-gram windows being processed. Couple this with the fact that

[6] While the training data can be kept, we do not consider this practical for actual deployment, especially if techniques like incremental and run-time training are used.

Bloom filters need a good source of hashes, and Anagram's largest computation overhead in both training and testing rapidly becomes the hash operation.

To reduce this computation overhead, we make use of a *cumulative universal hash function*. A cumulative hash function fulfills the requirement that $h\left(c\left(a,b\right)\right)=d\left(h\left(a\right),h\left(b\right)\right)$, where h represents our hash function, a and b are data (n-grams or fragments of them), c is a (bitwise) concatenation function, and d is a composition function. Given such a hash function, we can avoid re-computing hashes when sliding the n-gram window and when using different window sizes, e.g., if we've hashed a 5-gram and need to generate the hash of a 7-gram, we can just hash the incremental 2 grams and combine that with the 5-gram hash value. A class of universal hash functions known as H_3 [21] uses XOR as the composition function, which is very fast and lends itself well to our application. Thus, Anagram's modeling engine may perform fast enough to scale to very high bandwidth environments. Dharmapurikar, et. al. describe a similar technique [22] for static signature scanning in hardware.

2.3 Learning Issues

The huge feature space of higher order n-grams makes estimating the frequency distribution infeasible. In the case of representing distinct n-grams, we pose the following questions: how long do we need to train before deploying the sensor in detection mode? Further, how well can Anagram handle noisy training data since the model is binary-based?

2.3.1 When is the model well trained?

Since there are many distinct n-grams, many of which may take days or weeks to see, when can we say the model is well trained and ready to use? We first check the likelihood of seeing a new n-gram with additional training. Figure 2 shows the percentage of the *new* distinct n-grams out of every 10,000 content packets when we train for up to 500 hours of traffic data. The key observation is that during the initial training period, one should expect to see many distinct n-grams. Over time, however, fewer distinct "never before seen" n-grams should be observed. Hence, for a given value of n, a particular site should exhibit some *rate* of observing "new" distinct n-grams within its "normal" content flow. By estimating this rate, we can estimate how well the Anagram model has been trained. When the likelihood of seeing new n-grams in normal traffic is stable and low, the model is stable; we can then use the rate to help *detect* the attacks, as they should contain a higher percentage of unseen n-grams. Figure 3 plots the false positive rates of different models, varying in n-gram size and length of training time, when tested on the 72 hours of traffic immediately following the training data.

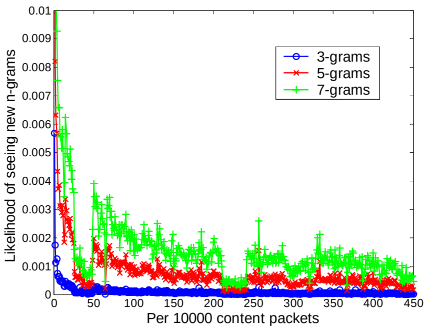

Fig. 3. The likelihood of seeing new n-grams as training time increases.

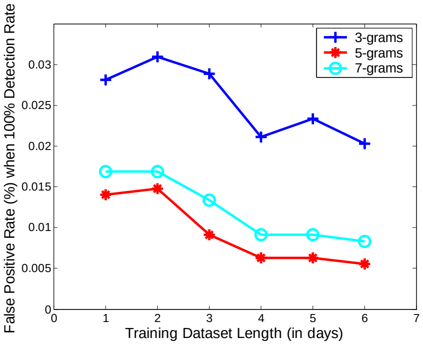

Fig. 4. False positive rate (with 100% detection rate) as training time increases.

From the above plots, we can see that as the training time increases, the false positive rate generally goes down as the model is more complete. After some point, e.g. 4 days' in figure 3, there is no further significant gain, and the FP rate is sufficiently low. Higher order n-grams need a longer training time to build a good model, so 7-grams display a worse result than 5-grams given the same amount of training. While the 3-gram model is likely more complete with the same amount of training, it scores significantly worse: 3-gram false positives do not stem from inadequate training, but rather because 3-grams are not long enough to distinguish malicious byte sequences from normal ones.[7]

In theory, Anagram should always improve with further training – if we can guarantee a clean training dataset, which is crucial for the binary-based approach. However, obtaining clean training data is not an easy task in practice. During our experiments, increased training eventually crosses a threshold where the false positive rate starts increasing, even if the training traffic has been filtered for all known attacks. The binary-based approach has significant advantages in speed and memory, but it's not tolerant of noisy training, and manual cleanup is infeasible for large amounts of training data. We therefore introduce *semi-supervised training* in the next section to help Anagram be more robust against noisy data.

2.3.2 Semi-supervised Learning

The binary-based approach is simple and memory efficient, but too sensitive to noisy training data. One form of supervised training is quite simple. We utilize the signature content of Snort rules obtained from [12] and (a collection of about 500) virus samples to precompute a known "bad content model". We build a *bad content Bloom filter* containing the n-grams that appear in the two collections, using all possible *n* that we may eventually train on (e.g., *n*=2~9 in our experiments). This model can be incrementally updated when new signatures have been released.

It's important to note, however, that signatures and viruses often contain some normal n-grams, e.g., the 'GET' keyword for HTTP exploits. To remove these normal n-grams, we can maintain a small, known-clean dataset used to exclude normal traffic

[7] We hypothesize, though, that as the number of new 3-grams converges very rapidly, that the rate of detection of unique 3-grams during testing may be a good sign of an attack. In general, anomalous content may be measurable as a function of the size of the gram detected that was not seen during training.

when generating the bad content BF. This helps to exclude, as a minimum, the most common normal n-grams from the bad content model.

In one experiment, we used 24 hours of clean traffic to filter out normal n-grams from the bad content BF. Figure 4 shows the distribution of normal packets and attack packets that match against the bad content model. The X axis represents the "matching score", the percentage of the n-grams of a packet that match the bad content model, while the Y axis shows the percentage of packets whose matching score falls within that score range. The difference between normal and attack packets is obvious; where the normal traffic barely matches any of the bad content model, attack packets have a much higher percentage, so that we can reliably apply the model for accurate detection. The resulting bad content BF contains approximately 46,000 Snort n-grams and 30 million virus n-grams (for *n*=2…9).

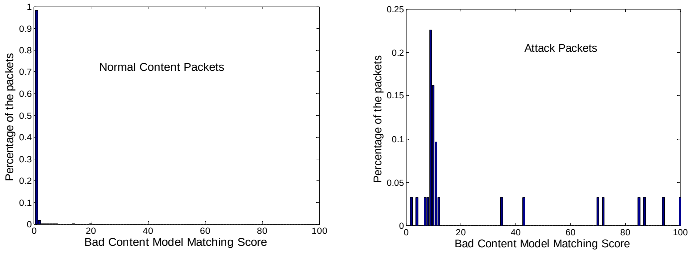

Fig. 5. Distribution of bad content scores for normal packets (left) and attack packets (right).

The "bad content model" is used during *both training and detection*. During training, the incoming data stream is first filtered through Snort to ensure it is free of known, old attacks. Packets are then compared against the bad content model; any n-gram that matches the bad content model is dropped. The whole packet is also dropped if it matches too many n-grams from the bad content model – as new attacks often reuse old exploit code – to avoid modeling new malicious n-grams. In our experiment, we established a 5% bad n-gram threshold before ignoring a training packet. While this is rather strict, ignoring a packet during training time is harmless as long as relatively few packets are dropped, as figure 4 shows.

During detection, if a never-before-seen n-gram also appears in the bad content model, its detection score is further weighted by a factor *t* over other malicious n-grams; in our experiment, we set *t* to 5. This enables us to further separate malicious packets from normal ones in order to achieve higher detection accuracy. To show the improvement we gain by using the bad content model, figure 5 compares the false positive rate before and after using it for different n-gram sizes on two datasets. The false positive rates are significantly reduced with the help of this bad content model.

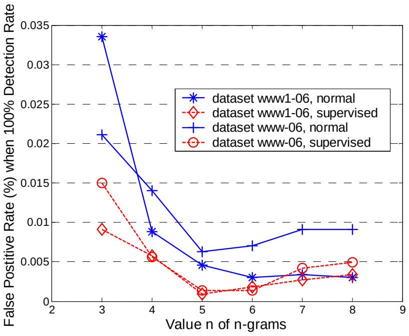

Fig. 6. The false positive rate (**with 100% detection rate**) for different n-grams, under both normal and semi-supervised training.

2.4 Signature Generation and Privacy-Preserving Cross-site Collaboration

One other substantial advantage of Anagram is its ability to generate robust signatures. When a suspicious packet is detected, the corresponding malicious n-grams are identified in the packet for free. These n-grams are good target for further analysis and signature generation. It's also particularly important to note that the order of the distinct anomalous n-grams is irrelevant, since it's just a set of n-grams. An attempt to avoid detection by an attacker by reordering their attack vector code will therefore fail.

Here we first revisit the php attack presented before, and then show the generated signatures by extracting the detected malicious n-grams, using 3-grams, 5-grams and 7-grams, respectively. Nonprintable characters are represented by ".", while "*" represents a wildcard for signature matching.

```
GET /modules/Forums/admin/admin_styles.phpadmin_styles.php?phpb
b_root_path=http://81.174.26.111/cmd.gif?&cmd=cd%20/tmp;wget%20
216.15.209.4/criman;chmod%20744%20criman;./criman;echo%20YYY;ec
ho|..HTTP/1.1.Host:.128.59.16.26.User-Agent:.Mozilla/4.0.(compa
tible;.MSIE.6.0;.Windows.NT.5.1;)..
```

N=3:

```
*?ph*bb_*//8*p;wg*n;c*n;./c*n;ec*0YYY;echo|H*26.U*1;).*
```

N=5:

```
*ums/ad*in/admin_sty*.phpadmin_sty*hp?phpbb_root_path=http://81
.174.26.111/cmd*cmd=cd%20/tmp;wget%20216*09.4/criman;chmod%2074
4%20criman;./criman;echo%20YYY;echo|  HTT*6.26.Use*5.1;)..*
```

N=7:

```
*dules/Forums/admin/admin_styles.phpadmin_styles.php?phpbb_root
_path=http://81.174.26.111/cmd.gi*?&cmd=cd%20/tmp;wget%20216.15
.209.4/criman;chmod%20744%20criman;./criman;echo%20YYY;echo|
HTTP/*59.16.26.User-*T 5.1;)..*
```

All of these signatures are quite accurate: no normal packet matches more than 5% of the malicious n-gram signatures of the attack. Generally, low order n-grams produces less optimal signatures, since relatively few n-grams will be detected as malicious and they are not long enough to capture useful information. 7-grams produce the best signature; in theory, the longer the n-gram the better the generated signature, higher training and false positive costs notwithstanding.

Besides the benefits of accurate signature generation, Anagram's models enable cross-site privacy-preserving information sharing by using Bloom Filters. In our previous PAYL work [11], we discussed the idea of cross-site collaboration and its use for zero-day worm detection and signature generation, followed by several approaches for alert information sharing. However, PAYL's correlation techniques traded accuracy for privacy.

Instead, Anagram can share Bloom Filters with no loss of accuracy. Suspect payloads are identified by the sensor, and the particular anomalous n-grams (of any size) are stored in a Bloom Filter.[8] The use of one-way hashes, combined with the large number of possible n-grams, makes reverse-engineering or brute forcing infeasible. By checking the n-grams of local alerts against the remote alert Bloom Filters, it's easy to tell how similar the alerts are to each other, and identify the common malicious n-grams that can be used to construct a candidate attack signature.

This is a substantial improvement over the previous approaches reported by several researchers whereby most common tokens, longest common substrings, and/or longest common subsequences are computed in order to identify common substrings between two or more suspect payloads. These computations intend to compute string signatures common across multiple instances of a polymorphic worm attack. Anagram essentially "pre-computes" common string subsequences via the common anomalous n-grams stored in the Bloom Filter, irrespective of the order of appearance of any of these anomalous strings, and thereby speeds up correlation.

To validate the accuracy of using a Bloom Filter of malicious n-grams for cross-site correlation, we collected alerts from the four previously-mentioned datasets and computed pairwise alert similarity scores, using the same worms for testing as before. The similarity score of a pair of alerts is defined as $2*N_c/(N_1+N_2)$, where N_c is the number of common n-grams, and N_1 and N_2 are the number of malicious n-grams of the different alerts, respectively. We used 5-grams in this experiment and set the malicious threshold to be zero, which means a packet will be flagged as an alert and correlated if there are one or more malicious n-grams. The results are summarized as:

1. Alerts from different datasets containing the same worm yield a similarity score greater than 0.9;
2. Alerts containing similar worms (e.g., CodeRed vs. CodeRed II) still score high, around 0.85, likely due to their similar code heritage;
3. Almost all other combinations yield a similarity score of less than 0.01, with less than .003% scoring more than 0.01 but less than 0.5.

Compared to our previous approaches for cross-site correlation, the Bloom filter based malicious n-grams can easily achieve better accuracy and privacy-preservation. This also illustrates the position independence of n-grams, as Anagram remains robust in detecting fragmented or reordered malcode, while ignoring non-malicious or padded material.

The BF representation *also* provides an efficient means of combining models computed from multiple sensors: one can simply bitwise AND the respective models to merge their contents—achieving performance scalability and faster learning at a remarkably low cost. We intend to further report on the effectiveness of these techniques, and their applicability to polymorphic worms, in an upcoming paper.

2.5 Implementation

The current Anagram prototype is implemented in approximately 3,000 lines of Java code. While we have recently built an optimized Bloom filter implementation using H_3, most of the models currently trained use an older, less optimized version based on SHA-

[8] The number of anomalous n-grams must also be transmitted, as it is used in the threshold logic to identify anomalous packets.

1. While we are still evaluating the faster hash algorithm's performance in training and detection, we expect to be able to handle a 100Mbps network subnet in real-time.[9] Anagram models take approximately 8MB of memory; Anagram, as a whole, uses approximately 20MB of memory when running (small enough to fit on a handheld device).

3 Randomized Models to thwart Mimicry Attacks

3.1 Anagram against Mimicry Attack

As mentioned earlier, mimicry attacks are perhaps the most significant threat to any anomaly detector, such as [1], which details a polymorphic mimicry worm targeting PAYL. This smart worm learns a site's normal traffic pattern by sniffing then blends the exploit packet with characters to simulate the normal byte frequency distribution to successfully evade PAYL's 1-gram models. Since this mimicry attack pads without considering the *sequence of bytes*, Anagram can easily detect any variants of the crafted attacks[10].

We adapted this worm to launch a mimicry attack against Anagram. Instead of padding the packet to simulate the byte frequency, we padded attack packets with normal *strings*; in this case, long URLs of the target website which should be, by definition, composed of normal n-grams. Although the anomaly scores are greatly reduced by this padding, the remaining portions of the crafted attack packets still have enough abnormal n-grams to be detected by Anagram. Besides the "sled", which provides the opportunity for crafted padding, the attack packet still requires a byte sequence for the polymorphic decryptor, the encrypted exploit, encoded attacks, and the embedded mapping table. Since the amount of space in each packet is limited, the mimicked worm content containing the exploit vector is purposely spread over a long series of fragmented packets. Thus, the worm is fragmented so that each packet on its own does not appear suspicious. This strategy is described in the aforementioned paper and is akin to a multi-partite attack strategy where the protocol processor assembles all of the distributed pieces necessary for the complete attack.

Using the blended polymorph worm engine, we generated different variants of the worm. The following table shows the maximum padding length of each version. Each cell in the top row contains a tuple (x, y), representing a variant sequence of *y* packets of *x* bytes each. The second row represents the maximum number of bytes that can be used for padding in each packet. It's obvious that there is a substantial chunk of packet that needs to be reserved for the exploit, where we conjecture malicious higher order n-grams will appear to encode the encrypted exploit code or the decryptor code.

Table 3. The padding length for a packet of different varieties of the mimicry attack

Version	418, 10	418, 100	730, 10	730, 100	1460, 10	1460, 100
Padding length	125	149	437	461	1167	1191

We tested Anagram over these modified mimicry attacks where the padding contained normal, non-malicious n-grams, and all of the attacks were successfully detected. This is the case since the crafted attack packets still require at least 15%-20% of the n-grams for code, which were detected as malicious. The false positive rates grows, however, as the packet length gets longer. The worst case for the (1460, 100) experiment yields a false positive rate around 0.1%. The semi-supervised learning strategy em-

[9] We also would expect to be able to give actual performance numbers by publication time.

[10] We are very grateful to Wenke Lee and his colleagues for providing the polymorph engine for use in our research.

ployed in Anagram using a model of “malicious n-grams” doesn’t help here since this mimicry worm uses encryption and data encoding to obfuscate its content.

This experiment demonstrates that Anagram raises the bar for attackers making mimicry attacks harder since now the attackers have the task of carefully crafting the *entire* packet to exhibit a normal content distribution. Further effort is required by mimicry attacks to encode the attack vectors or code in a proper way that appears as normal high order n-grams. Without knowing exactly which value of *n,* the size of the modeled grams, they should plan for, the problem becomes even harder. We take this uncertainty and extend it in the next section for a more thorough strategy to thwart mimicry attacks.

3.2 Randomization

The general idea of payload-based mimicry attack is simply to evade detection by crafting small pieces of exploit code with a large amount of “normal” padding data to make the whole packet look normal. But as we’ve seen in the example above, no matter what techniques are used for padding, there has to be some non-padded “exposed” sections of data to decode the exploit of the target vulnerability. The attacker has to determine *where* to pad with normal data, and where to hide the exploit code. Without knowing exactly what portions of the packet are tested by the detector, the task is complicated, and possibly reduced to guessing, even if the attacker knew what padding would be considered normal.

We first discuss the notion of randomized modeling. Instead of modeling and testing the whole packet payload, we randomly partition packets into several (possibly interleaved) substrings or subsequences $S_1,S_2,\ldots,S_N$, and model and test each of them separately. Since the partition is randomly chosen by each sensor instance and kept secret, we assume the attackers cannot gain this information before they compromise the machine. The attempted mimicry attack would be thwarted since the attacker would have no means of knowing precisely how a remote payload anomaly sensor at its target location has chosen to partition the space of the data flow. The attackers could only succeed if they can craft an attack vector ensuring that the data is normal with respect to any randomly selected portion of a packet; this makes the attacker’s tasks *much* harder than before. This technique can be generally applied to any content anomaly detector.

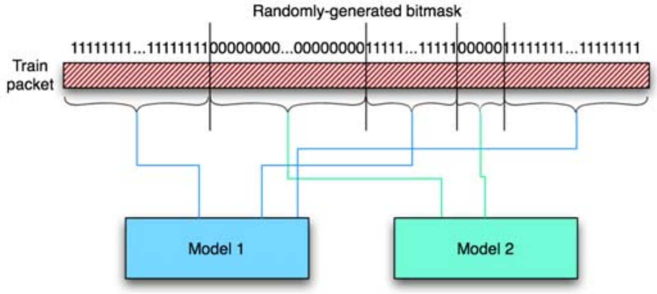

Fig. 7. Randomized modeling.

We performed experiments using randomized modeling for PAYL, where we computed multiple models based on different secret partitions of the data stream. Although the strategy successfully thwarted mimicry attack, it substantially increased the overhead of the sensor. A future report will detail those experiments.

Alternatively, we can employ *randomized testing*, which does not incur any substantial overhead. Instead of testing and scoring the whole packet payload, we randomly partition packets into several (possibly interleaved) substrings or subsequences $S_1, S_2, \ldots, S_N$, and test each of them separately against the same single normal BF model and the single bad content model. Since the partition is randomly chosen by each sensor instance and kept secret, we assume the attackers cannot gain this information before they compromise the machine. The attackers could only succeed if they can craft an attack vector ensuring that the data is normal with respect to any randomly selected portion of a packet; this makes the attacker's tasks *much* harder than before. This technique can be generally applied to any content anomaly detector.

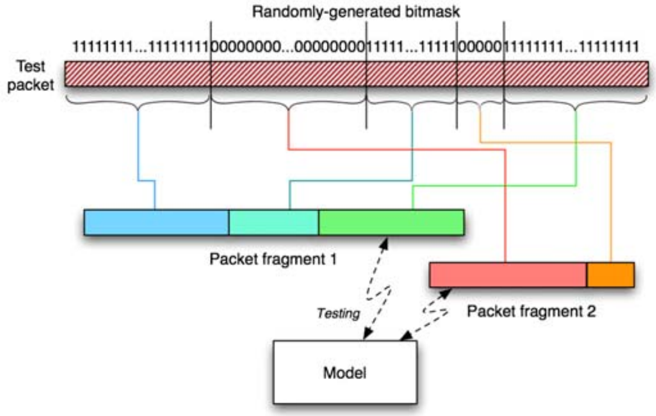

Fig. 8. Randomized testing.

To demonstrate the effectiveness of this counter-evasion tactic, we developed a simple randomization framework for Anagram. We generate a random binary mask with some length *l* (say, 100 bytes), and repeatedly apply this mask to the contents of each packet to generate test partitions. The mask corresponds to subsequences of contiguous bytes in the packet tested against the model; each is scored separately. The sequence of bytes appearing at locations where the mask had bits set to 1 are tested separately from those byte sequences where the mask had zero bits, randomly separating it into two non-overlapping parts. The packet anomaly score is adapted from section 2.1 to maximize the score over all partitions, i.e. $Score = \max\left(N_{i_{new}} / T_i\right)$, where $N_{i_{new}}$ and T_i are the number of new and total n-grams in partition *i*, respectively. This can easily be extended to overlapping regions of the packet, and more than just two randomly chosen portions.

There are several issues to consider. First, we want to optimize binary mask generation. While the mask can be a purely random binary string, we may then lose information about sequences of bytes. Since Anagram models n-grams, it's not surprising that this strategy performs poorly. Instead, we randomize the mask using a "chunked" strategy. Any string of contiguous 0's or 1's in the mask must be at least 10 bits long (corresponding to 10 contiguous bytes in a partition), enabling us to preserve most of the n-gram information for testing. The results achieved for this strategy are far better than a random binary mask.

Another observation is that the length of the randomly chosen partitions is best balanced, which means the two parts should have roughly the same size, i.e. equal numbers of 1's and 0's. The false positive rate is usually much higher when the partitions have extremely unbalanced lengths; for example, a partition where one fragment is 10% of the total length and the other is 90% produces a poor false positive rate.

For the following results, we use this “chunk-based binary mask strategy” and guarantee that one partition of the packet datagram is at most double the size of the other one. Again, we measure the false positive rate when we achieve *100%* detection rate for our test traces. For each size n-gram, we repeated the experiment 10 times to show the average (the middle plot) and standard deviation (the vertical bars), both for the unsupervised learning (left plot) and semi-supervised learning (right plot) using the malicious n-gram content model. The experiment was performed on dataset *www1-06*, trained for 90 hours and tested on the following 72 hours of traffic.

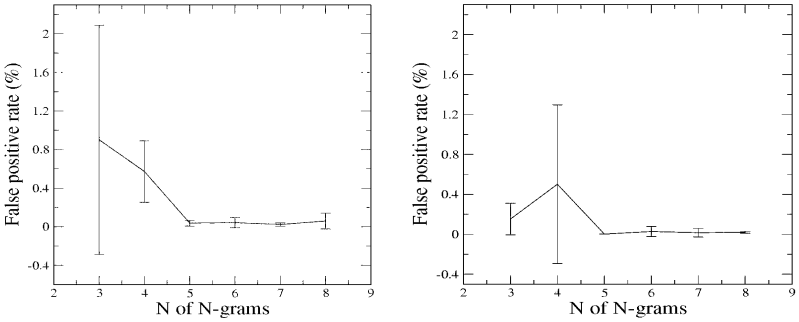

Fig. 9. The average false positive rate and standard deviation with 100% detection rate for randomized testing, with normal training (left) and semi-supervised training (right).

On average, the randomized models produce comparable false positive rates to the non-randomized approach, especially when using semi-supervised learning. The lower order n-grams are more sensitive to partitioning, so they exhibit a high standard deviation, while higher order n-gram models are relatively stable. Further research needs to be done to determine a good partitioning approach that can reduce the deviation while keeping the essential randomization strategy intact. We believe randomization is the correct direction, and a valuable step toward complicating the task for the mimicry attacker.

3.3 Threshold reduction and “extreme” padding

In the experiments we reported above, we noticed that the randomized models’ false positive rates, while comparable on average, exhibit a higher variance than the case where models are not randomized. Consider an extreme mimicry attack: suppose an attacker crafts packets with *only one instruction* per packet, and pads the rest with normal data. If a 100% detection rate is desirable, lowering the score threshold to some minimum nonzero value ε might be the only way to achieve this goal. Unsurprisingly, this approach corresponds to a direct increase in the false positive rate, which may vary between 10% to 25% of all packets, depending on the n-gram sizes chosen and the amount of training of the model.

Such a false positive rate may be viewed as impractical, rendering the sensor useless. *We believe this is wrong.* The false positive rate is not always the right metric – especially if Anagram is used in conjunction with other sensors. For example, Anagram may be used to *shunt* traffic to a host-based, heavily instrumented *shadow server* used as a detector; in other words, we do not generate Anagram alarms to drop traffic or otherwise mitigate against (possibly false) attacks, but rather we may validate whether traffic is indeed an attack by making use of a host-based sensor system. If we can shunt, say 25% of the traffic for inspection by an (expensive) host-based system, while leaving the remaining normal traffic unabated to be processed by the operational servers, the proposed host-based shadow servers can amortize their overhead costs far more economically. Thus, the false positive rate is not as important as the true positive rate, and false positives do not cause harm. We describe this approach in greater detail in section 4.

4 Adaptive Learning

4.1 Training Attacks versus Mimicry Attacks

We distinguish between training attacks and mimicry attacks: A *mimicry attack* is the willful attempt to craft and shape an attack vector to look normal with respect to a model computed by an anomaly detector. The attacker would need to know the modeling algorithm used by the sensor and the normal data it was trained on. The polymorphic blended attack engine discussed in Section 3 assumes to know both by sniffing an environment's normal data (an open question is whether the normal data of one site produces sufficiently similar models to other sites that are targeted by a mimicry attack). Alternatively, a *training attack* is one whereby the attacker sends a stream of data incrementally or continuously distant from the normal data at a target site in order to influence the anomaly detector to model data consistent with the attack vector. Attack packets would appear normal since they were modeled. This type of attack would succeed if the attacker were lucky enough to send the stream of data *while* the anomaly detector was in training mode. Furthermore, the attacker would presume that the stream of "malicious training data" would go *unnoticed* by the sensor while it was training.

We presented the concept of randomization in order to thwart mimicry attack. Even if the attacker knew the normal data distribution, the attacker would not know the actual model used by the sensor. However, we have not addressed training attacks. [23] explores this problem and suggests several theoretical defenses. For example, Anagram's semi-supervised learning mechanism can help protect the model if learning attacks recycle old exploit code. However, if the learning attack does not contain any known bad n-grams, Anagram cannot detect it by itself. We conjecture that the coupling of the training sensor with an "oracle" that informs the sensor of whether or not the data it is modeling is truly normal can thwart such training attacks. For example, if the attacker sends packets that do not exploit a server vulnerability, but produces an error response, this should be noticed by the sensor in training mode; we discuss this further in the next section. Such feedback-based learning does not address all cases, e.g., a learning attack embedded into a HTTP POST payload, which would generate a legitimate server response. Randomization may also be valuable for learning attacks; we leave the exploration of such defense mechanisms for future research.

4.2 Feedback-based learning and filtering using instrumented *shadow servers*

Host-based fault detection and patch generation techniques (such as StackGuard/MemGuard [24], STEM [25], DYBOC [26], and many others) hold significant promise in improving worm and attack detection, but at the cost of significant computational overhead on the host. The performance hit on a server could render such technologies of limited operational value. For instance, STEM [25] imposes a 200% overhead for an entirely-instrumented application. DYBOC [26] is more proactive and designed to be deployed on production servers to provide faster response, but still imposes at least a 20% overhead on practical web servers. If one can find a means of reducing the cost of this overhead, the technology will have a far greater value to a larger market.

We envision an architecture consisting of both *production* servers and an instrumented *shadow* server, with the latter executing both valid and malicious requests securely but with significant overhead. A *network anomaly flow classifier* is placed in front of these pools and *shunts* traffic based on the anomaly content in incoming requests, as shown in figure 10.

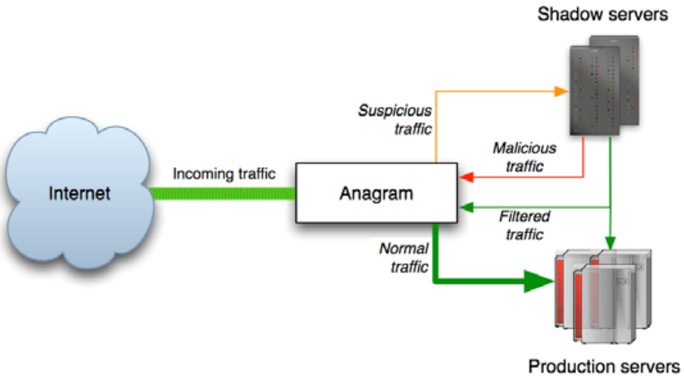

Fig. 10. Shadow server architecture.

In order for the flow classifier to be appropriate for this architecture, we need to ensure that *no* malicious requests are sent to the production pool, as those machines may be potentially vulnerable to zero-day attacks. *It is acceptable* if a small fraction of the traffic deemed as false positives are shunted to the shadow server, because this server will indeed serve the requests, but with a greater latency. Nothing has been lost, but only some amount of response time for a minority portion of the traffic flow. In other words, an anomaly detector that wants to act as this classifier should have a 100% true positive detection rate and a reasonably low false positive rate. We can characterize the latency in such an architecture as $l' = \left(l \times \left(1 - fp\right)\right) + \left(l \times o_s \times fp\right)$, where l is the standard latency of a service, o_s is the shadow server overhead, and *fp* is the false positive rate. If we want to target a maximum latency increase $l' - l$ of 1%, given a 20% overhead in using the shadow server, the false positive rate can be as high as approximately 10%. As we described in section 3.3, we believe this order-of-magnitude is achievable, along with a 100% detection rate, by setting the threshold to some low ε. This places network content flow anomaly detectors in a completely new light. Zero false positive rates are simply not the right objective. Furthermore, a symbiotic relationship may emerge over time. False positives validated by the shadow server serve as training data to improve the accuracy of the anomaly detector. In time, the false positive rate would be decrease and less traffic would be served by the shadow server; we discuss this in the next section.

4.2.1 Adaptive model training of Anagram with shadow servers

Anagram's detection model assumes no noise in the model, and uses semi-supervised training (section 2.3.2) to ensure clean training traffic. This approach can be enhanced with a hybrid shadow server architecture:

1. Use the shadow server as a training supervisor. This entails an initial *passive* Anagram deployment, and a "slow training mechanism" whereby the shadow server is initially sent all requests, and *only requests generating a normal response* are then

sent to Anagram for training. After a sufficient amount of training has been done, Anagram can then be put into an active deployment.

2. Use a short training time, and use the shadow server as a false positive feedback mechanism. In this scenario, Anagram only trains on a small fraction of traffic assumed to be good and is then immediately deployed. While the false positive rate will be higher, Anagram can watch for normal responses from the shadow server and can then include the appropriate n-grams of the original request in its model of "normal traffic". This can incrementally reduce the false positive rate as the system continues processing requests.

Of course, a hybrid approach can also be employed; for example, Anagram could be deployed with no model, process 100% as false positives, and use feedback as an incremental learning architecture. We performed some early experiments with feedback learning using the PAYL sensor in [27], and Anagram has recently been used as a sensor in [28]. We intend to continue integration experiments using Anagram, and will report our results in a future paper.

5 Related Work

In addition to the works cited in the introduction, we further discuss research in anomaly detection, signature generation, polymorphic worms, and mimicry and learning attacks.

5.1 Anomaly Detectors and Signature Generators

Early intrusion anomaly sensors focused system calls. Forrest [16]'s "foreign sequences of system calls" in a binary-based anomaly detector is quite similar to the modeling implemented in Anagram. Tan and Maxion [17] shows why Forrest's work produced optimal results when the size of the token window was fixed at 6 (essentially a 6-gram). In Anagram, no fixed sized window is needed; one may model a mixture of n-grams. Forrest's grams were sequences of tokens each representing a unique system function, whereas Anagram models n-grams of byte values.

Many researchers have considered the use of packet flows and/or content analysis for anomaly detection. Honeycomb [29] is a host-based IDS that automatically creates signatures by applying longest common substring (LCS) on malicious traffic captured by a honeypot targeting dark space. Computed substrings are used as candidate worm signatures. Similarly, EarlyBird [30] uses Rabin fingerprints to find the most frequent substrings for signatures. Unlike Honeycomb (and PAYL), Anagram computes distinct n-grams from packets and compares the n-gram set against those in its model; this is a linear-time operation, unlike polynomial-time LCS.

Polygraph [31] extends the work done in Autograph [32]; both are signature generators that assume traffic is separated into two flow pools, one with suspicious scanning traffic and a one with non-suspicious traffic. Instead of assuming signatures are contiguous, like Autograph, Polygraph allows a signature composed multiple noncontiguous substrings (*tokens*), particularly to accommodate polymorphic worms. Tokens may be generated as a set (of which all must be present), as an ordered sequence, or as a probabilistic set (Bayes signature). Like Polygraph, Anagram is capable of identifying multiple "tokens". However, Anagram's design also does not assume an external flow classifier, being one itself. A more general discussion of related work in the area of anomaly detection can be found in [10].

Shield [33] provides vulnerability signatures instead of string-oriented content signatures, and blocks attacks that exploit that vulnerability. A "shield" is manually specified for a vulnerability identified in some network available code; the time lag to specify, test and deploy shields from the moment the vulnerability is identified favors the worm writer, not the defenders. COVERS [34] analyzes attack-triggered memory errors in C/C++ programs and develops structural memory signatures; this is a primarily host-specific approach, while Anagram focuses on network traffic.

SigFree [9] uses a different approach, focusing on generic code detection; as its name implies, it does not rely on signatures, preferring to disassemble instruction sequences and identify, using data flow anomaly detection, if requests contain sufficient code to merit them as being suspicious. Anagram does not explicitly differentiate between code and data, although it is often able to do so based on training. Additionally, Anagram monitors content flows, not just requests, and can apply to a broader variety of protocols.

5.2 Polymorphic Worms, Mimicry and Learning Attacks

Polymorphic viruses are nothing new; "1260" and the "Dark Avenger Mutation Engine" were considered the first two polymorphic virus engines, written in the early 90s [35]. However, early work focused on making it more difficult for detection by COTS signature scanners, would be easily detected by an anomaly detector as they contain significantly different byte distributions than non-malicious code, and were primarily targeted for manual execution and so did not incorporate exploit mechanisms like common Internet worms. Polymorphic worms with vulnerability-exploiting shellcode, e.g., ADMmutate [36] and CLET [37], do support exploit vectors and are primarily designed to fool signature-based IDSes. CLET does feature a form of padding, which they call *cramming*, to defeat simple anomaly detectors. However, cram bytes are derived from a static source, i.e. instructions in a file included with the CLET distribution; while this may be customized to approach a general mimicry attack, it must be done by hand.

The notion of a *mimicry attack* on an anomaly detection system was first introduced in 2001 by Wagner and Dean [38], but initial efforts to generate mimicry attacks, including [39] and [40], focused on system-call anomaly detection. With the advent of effective network payload-based anomaly detection techniques, researchers have begun building "smart worms" that employ a combination of polymorphism and mimicry attack mechanisms. Kolesnikov, Dagon and Lee [1] built a worm specifically designed to target network anomaly detection approaches, including PAYL. They use a number of techniques, including polymorphic decryption, normal traffic profiling and blending, and splitting to effectively defeat PAYL and several other IDSes.

Defeating *learning attacks* is also current research; [23] discusses the problem for anomaly detectors from a theoretical perspective, categorizes different types of learning attacks (e.g., causative vs. exploratory, etc.) and speculates as to several possible solutions. Anagram implements some of these techniques, and its use of randomization, hiding key parameters of the model secret from the attacker, may be extensible to learning. Our ongoing work includes exploring several other strategies, including the randomization of n-gram sizes, and various strategies to test whether an attacker is polluting learning traffic at given points in time.

6 Conclusion

Anagram is an anomaly detector based upon *n-gram analysis* using a *binary-based modeling* technique. The use of *randomization* makes the sensor resistant to mimicry attack. Anagram's models use *Bloom filters* for compactness and performance. Our tests suggest Anagram has less than a .01% false positive rate along with a 100% detection rate for a variety of worms and viruses detected in traces of our local network traffic.

Anagram's use of Bloom filters also enables effective *privacy-preserving cross-site correlation* and *signature generation*.

Anagram detects existing mimicry attacks, including those targeted at our previous anomaly detection sensor, PAYL, and we speculate that Anagram will be robust to future attacks as well. We also discuss approaches to effectively learn Anagram models, including the use of a *bad content Bloom filter* and *instrumented shadow servers*. For the latter case, we believe Anagram can act as an effective *network anomaly flow classifier* to mitigate host instrumentation overhead and make these tools effective for practical deployment.

There are a number of interesting venues for future research. We intend to build an integrated instrumented shadow server architecture utilizing Anagram to collect statistics on performance and modeling accuracy. Another open area of research is to make binary-based modeling more robust against learning attacks. We would also like to compare Anagram's performance to other proposed content-based anomaly sensors.[11] Finally, we intend a practical deployment of a multiple-site correlation effort and gauge Anagram's performance in helping to identify broad zero-day worms or targeted attacks while maintaining full privacy.

Acknowledgements

We would like to thank Gabriela Cretu, Wei-Jen Li, Michael Locasto, Angelos Stavrou, and Angelos Keromytis for their suggestions and advice during our collaboration, and Panagiotis Manolios and Peter Dillinger for their suggestions in Bloom filter design.

References

1. Kolesnikov, O., D. Dagon, and W. Lee, *Advanced Polymorphic Worms: Evading IDS by Blending in with Normal Traffic*, in *USENIX Security Symposium*. 2006, Georgia Tech: Vancouver, BC, Canada.
2. Moore, D., et al. *Internet Quarantine: Requirements for Containing Self-Propagating Code*. in *INFOCOM*. 2003.
3. Staniford-Chen, S., V. Paxson, and N. Weaver. *How to 0wn the Internet in Your Spare Time*. in *USENIX Security*. 2002.
4. Christodorescu, M. and S. Jha. *Static Analysis of Executables to Detect Malicious Patterns*. in *USENIX Security Symposium*. 2003. Washington, D.C.
5. Vargiya, R. and P. Chan. *Boundary Detection in Tokenizing Network Application Payload for Anomaly Detection*. in *ICDM Workshop on Data Mining for Computer Security (DMSEC)*. 2003. Melbourne, FL.
6. Kruegel, C., et al. *Polymorphic Worm Detection Using Structural Information of Executables*. in *Symposium on Recent Advances in Intrusion Detection*. 2005. Seattle, WA.
7. Sekar, R., et al. *Specification-based Anomaly Detection: A New Approach for Detecting Network Intrusions*. in *ACM Conference on Computer and Communications Security*. 2002. Washington, D.C.
8. Kruegel, C., T. Toth, and E. Kirda. *Service Specific Anomaly Detection for Network Intrusion Detection*. in *Symposium on Applied Computing (SAC)*. 2002. Madrid, Spain.
9. Wang, X., et al. *SigFree: A Signature-free Buffer Overflow Attack Blocker*. in *USENIX Security*. 2006. Boston, MA.
10. Wang, K. and S.J. Stolfo. *Anomalous Payload-based Network Intrusion Detection*. in *Symposium on Recent Advances in Intrusion Detection*. 2004. Sophia Antipolis, France.

[11] At the time of writing this paper, no other sensors have been made available to us from other researchers. We would be delighted to collaborate with other research groups to perform these comparative evaluations.

11. Wang, K., G. Cretu, and S.J. Stolfo. *Anomalous Payload-based Worm Detection and Signature Generation*. in *Symposium on Recent Advances in Intrusion Detection*. 2005. Seattle, WA.
12. SourceFire Inc. *Snort rulesets*. 2006 [cited 2006 April 4]; Available from: http://www.snort.org/pub-bin/downloads.cgi.
13. Locasto, M.E., S. Sidiroglou, and A.D. Keromytis. *Application Communities: Using Monoculture for Dependability*. in *HotDep*. 2005.
14. Locasto, M.E., S. Sidiroglou, and A.D. Keromytis. *Software Self-Healing Using Collaborative Application Communities*. in *Internet Society (ISOC) Symposium on Network and Distributed Systems Security*. 2006. San Diego, CA.
15. Marceau, C. *Characterizing the Behavior of a Program Using Multiple-Length N-grams*. in *New Security Paradigms Workshop*. 2000. Cork, Ireland.
16. Forrest, S., et al. *A Sense of Self for Unix Processes*. in *IEEE Symposium on Security and Privacy*. 1996.
17. Tan, K.M.C. and R.A. Maxion. *Why 6? Defining the Operational Limits of stide, an Anomaly-Based Intrusion Detector*. in *IEEE Symposium on Security and Privacy*. 2002. Berkeley, CA.
18. Crandall, J.R., et al. *On Deriving Unknown Vulnerabilities from Zero-Day Polymorphic and Metamorphic Worm Exploits*. in *ACM Conference on Computer and Communications Security*. 2005. Alexandria, VA.
19. Bloom, B.H., *Space/time trade-offs in Hash Coding with Allowable Errors.* Communications of the ACM, 1970. **13**(7): p. 422-426.
20. Naor, M. and M. Yung. *Universal One-Way Hash Functions and their Cryptographic Applications*. in *ACM Symposium on Theory of Computing*. 1989. Seattle, WA.
21. Ramakrishna, M.V., E. Fu, and E. Bahcekapili. *A Performance Study of Hashing Functions*. 1994.
22. Dharmapurikar, S., et al. *Deep Packet Inspection using Parallel Bloom Filters*. in *IEEE Symposium on High Performance Interconnects (HOTI)*. 2003.
23. Barreno, M., et al. *Can Machine Learning Be Secure?* in *ASIACCS*. 2006.
24. Cowan, C., et al. *StackGuard: Automatic Adaptive Detection and Prevention of Buffer-Overflow Attacks*. in *USENIX Security Symposium*. 1998. San Antonio, TX.
25. Sidiroglou, S., et al. *Building a Reactive Immune System for Software Services*. in *USENIX*. 2005. Anaheim, CA.
26. Sidiroglou, S., G. Giovanidis, and A.D. Keromytis. *A Dynamic Mechanism for Recovering from Buffer Overflow Attacks*. in *8th Information Security Conference*. 2005. Singapore.
27. Locasto, M.E., et al. *FLIPS: Hybrid Adaptive Intrusion Prevention*. in *Symposium on Recent Advances in Intrusion Detection*. 2005. Seattle, WA.
28. Locasto, M.E., M. Burnside, and A.D. Keromytis, *Bloodhound: Searching Out Malicious Input in Network Flows for Automatic Repair Validation*. 2006, Columbia University Department of Computer Science: New York, NY.
29. Kreibich, C. and J. Crowcroft. *Honeycomb - Creating Intrusion Detection Signatures Using Honeypots*. in *ACM Workshop on Hot Topics in Networks*. 2003. Boston, MA.
30. Singh, S., et al. *The EarlyBird System for Real-Time Detection of Unknown Worms*. in *ACM Workshop on Hot Topics in Networks*. 2003. Boston, MA.
31. Newsome, J., B. Karp, and D. Song. *Polygraph: Automatically Generating Signatures for Polymorphic Worms*. in *IEEE Security & Privacy*. 2005. Oakland, CA.
32. Kim, H.-A. and B. Karp. *Autograph: Toward Automated, Distributed Worm Signature Detection*. in *USENIX Security Symposium*. 2004. San Diego, CA.
33. Wang, H.J., et al. *Shield: Vulnerability-Driven Network Filters for Preventing Known Vulnerability Exploits*. in *ACM SIGCOMM*. 2004.
34. Liang, Z. and R. Sekar. *Fast and Automated Generation of Attack Signatures: A Basis for Building Self-Protecing Servers*. in *ACM Conference on Computer and Communications Security*. 2005. Alexandria, VA.
35. Szor, P. and P. Ferrie. *Hunting for Metamorphic*. in *Virus Bulletin Conference*. 2001. Prague, Czech Republic.

36. K2. *ADMmutate*. 2001 [cited 2006 March 29]; Available from: http://www.ktwo.ca/security.html.
37. Detristan, T., et al. *Polymorphic Shellcode Engine Using Spectrum Analysis*. Phrack 2003 [cited 2006 March 28]; Available from: http://www.phrack.org/show.php?p=61&a=9.
38. Wagner, D. and D. Dean. *Intrusion Detection via Static Analysis*. in *IEEE Security & Privacy*. 2001. Oakland, CA.
39. Wagner, D. and P. Soto. *Mimicry Attacks on Host-Based Intrusion Detection Systems*. in *ACM CCS*. 2002.
40. Tan, K.M.C., K.S. Killourhy, and R.A. Maxion. *Undermining an Anomaly-Based Intrusion Detection System Using Common Exploits*. in *Symposium on Recent Advances in Intrusion Detection*. 2002. Zurich, Switzerland.