Pre-print of article that appeared at CISS 2006.

The published article can be accessed from:
http://ieeexplore.ieee.org/search/freesrchabstract.jsp?tp=&arnumber=4068038

The Strength of Syntax Based Approaches to Dynamic Network Intrusion Detection

W. Scheirer, M. Chuah
{wjs3, chuah}@cse.lehigh.edu
Department of Computer Science and Engineering
Lehigh University

Abstract - In this paper, we investigate three *syntax* based, sliding window schemes for automatic intrusion detection. The first method, the fixed partition sliding window scheme (FPSW), uses a fixed window size and a one-byte sliding window. The second method, referred to as variable-length partition sliding window (VPSW), uses a variable length window with a predetermined breakmark. The third method, referred to as variable-length partition with multiple breakmarks (VPMB), is similar to VPSW except that multiple breakmarks are used. The results indicate that while the FPSW and VPSW methods are effective for detecting worms with mild changes in the worm code contents, VPMB is suitable for detecting fully polymorphic worms.

1. Introduction

The intrusion detection system (IDS) has become extremely important in recent years to combat a rapid increase in malicious software. On the network, modern worms can spread very quickly and widely, thus automatic response is required to contain outbreaks. Two of the most popular signature based IDS systems are Snort [1] and Bro [2]. Each was designed to be more adaptable to emerging threats, but only through human intervention.

Computers allow us to represent the same data in variety of different ways. This useful flexibility turns against us in IDS, where reliable pattern matching is desirable. Figure 1 is a simple example of Unicode being used to encode a single character to thwart a static IDS signature. IDS authors responded to such encoding attacks by including decoders for popular methods. This, however, proves to be inadequate, as a multitude of clever encoding schemes exist, allowing an attacker to craft special URLs with whitespace, directory traversals (i.e ../../../), and local directory access (i.e. ./././) with ease.

Original: Get NULL.idaXXXXXXXXXXX

Encoding: Get NULL.id%u0061XXXXXXXXXXX

Figure 1. unicode encoding of 'a' in HTTP GET request

Moreover, attackers are certainly not limited to attacks with ASCII features. Malicious binary code may be encrypted with a unique key, completely obscuring its purpose. This form of obfuscation is a type of *Polymorphism*. It is also possible to alter the very structure of binary code, through code transposition, equivalent instruction substitution, *jump* insertion, NOP insertion, garbage instruction insertion, and register reassignment. This is known as *Metamorphism*. Thus, we are presented with the key question this paper attempts to address: How can we reliably detect threats on the network in the face of such dynamic malicious traffic?

In order to answer this question, we will investigate the syntax based, sliding window approach to dynamic IDS, and evaluate three competing schemes. The rest of the paper is organized as follows: In Section 2, we describe some related work and discuss how several particular works motivate this research. In Section 3, we describe the motivation and design behind each of the sliding window methods. With this knowledge, we proceed to Section 4, where each method is evaluated using an extensive set of real network traffic. Finally, we summarize our findings and discuss some future work that we intend to explore in Section 5.

2. Related Work

In [3], the authors present the case for collaborative intrusion detection system where intrusion detection nodes cooperate to determine if a network attack is taking place and take corrective actions if it does. Others have sought to use statistical approaches to detect worm outbreaks. In [4], the authors propose a method to identify a worm victim by observing if the number of scans per second it performs exceeds a certain threshold. The numbers of worm victims observed in successive windows are then compared to the numbers predicted using a typical worm spread model and if they match, then a worm outbreak is declared.

Sliding window schemes [5] [6] are based on the premise that some portion of malicious code will inevitably be invariant, despite attempts at obscuring its true nature to avoid detection. Autograph [5] was the first system of this type to emerge, and based its detection on content-based signature generation driven by breakmark delimited windows. As a follow-up to Autograph, Earlybird [6] used the same content-based signature methodology, but improved the architecture as a NIDS. Refinement

takes place in what the authors term "content sifting," whereby precise signatures are generated in a more efficient method (through the use of optimized data structures) than Autograph.

3. Sliding-Window Schemes

In this section, we describe a prototype system that has been built to automatically generate new worm signatures. This approach builds upon the previous work done in [5] and [6]. The approaches in [5] and [6] are based on the facts that a certain portion of the content in a worm packet is invariant and that the spreading of a worm is atypical of Internet applications. In [5], a variable-length sliding window of size *b* bytes is used to generate Rabin fingerprints of suspicious worm packets. The Rabin fingerprint [7] [8] is a one-way hashing algorithm that produces a 64-bit digest output. Such fingerprints allow us to do clustering easily. The payload of the packet is partitioned into multiple chunks when a chosen breakmark is detected. In [6], a fixed-length sliding window is used instead. However, there is no comparison of whether a fixed-length partition or a variable length partition is better in [5] or [6]. In this work, we compare both the fixed-length and variable-length partition approach. In addition, the work in [5] is extended by using a set of breakmarks rather than just a single breakmark.

3.1 Method 1 - Fixed Partition Sliding Window (FPSW) Scheme

Method 1, the FPSW scheme, incorporates a fixed window size and a one-byte window sliding. The premise is simple - a series (perhaps large, depending on window size) of fingerprints is generated as the window slides down the payload of a packet. We will see common signatures between different packet payloads if their contents are identical or similar. If the window size is small enough, common data portions can be isolated, despite the variation in the overall payloads. This is useful for the dynamic detection of new worm variants (e.g. W32.Blaster versus W32.Blaster.H). Figure 2 shows the operation of the sliding window using FPSW. The segments in *f*0, *f*1, and *f*2 will all be fingerprinted.

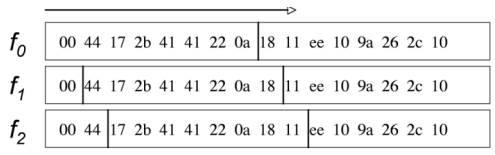

Figure 2. Three instances of an 8-byte sliding window, beginning at '00' for the FPSW scheme

One important decision related to this approach is choosing a proper window size. If the window size is very small (just a few bytes), the false positive rates will be higher. Certain short sequences are bound to appear in benign traffic as well as in malicious code. For example, "GET /" is typically at the beginning of a basic web request, but could also be followed by malicious exploit code. A 5-byte window would match both to the same fingerprint. In addition, the amount of signatures generated is always related to the window size. Smaller windows will produce more fingerprints, thus placing a higher burden on storing and searching.

3.2 Method 2 - Variable-length Partition Sliding Window (VPSW) Scheme

Method two, the variable-length partition sliding window scheme (VPSW), incorporates a one-byte sliding window approach until a predetermined breakmark is reached. This is similar to FPSW, except now the window

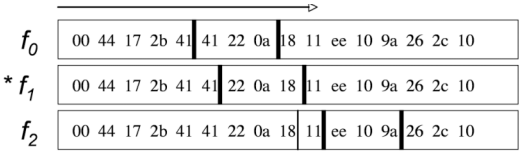

Figure 3. Example of VPSW operation with a breakmark of "22 0a 18"

grows until it reaches a pre-determined breakmark. When a breakmark match occurs, the fingerprint is generated, and the window begins to grow again from the stopped position. VPSW is similar to the scheme introduced in [5] except that different breakmarks can be configured. Figure 2.4 shows the operation of VPSW's sliding window. In this example, our breakmark is "22 0a 18". *f*0 displays a sliding window in progress, with the last three bytes ('41 22 0a') representing the break-mark match attempt. It fails, so the window is shifted over one byte. At *f*1, the last three bytes of the window match the breakmark, causing a fingerprint to be generated. Following this, the window begins again in *f*2, one byte directly after the previous breakmark match.

In practice, this method turns out to be a poor performer. The variance of data before the breakmark makes it much harder to produce repeatable signatures. Certain worms such as Nimda contain dynamic content, while different

versions of the same code also have variations - all of which may occur before a good static breakmark. Another problem is in choosing an appropriate breakmark. Selecting a series of NOP instructions (0x90 for x86) is a good choice for detecting specific types of overflow exploits, but certainly it does not cover every type of malicious traffic. Thus, much effort is required for selecting appropriate breakmarks. Despite these shortcomings, it is important that we introduce the VPSW scheme because a slight variation of this method turns out to be excellent for detecting polymorphic exploits. This third method is explained next.

3.3 Method 3 - Variable-length Partition with Multiple Breakmarks Scheme (VPMB)

Method three, the variable-length partition with multiple break-marks scheme (VPMB), is identical to Method 2, except that the breakmark is tuned to match a series of NOP-like instructions. In a polymorphic exploit, we often see a static region initiating a request (for example, a web based exploit may begin with a normal HTTP GET request), followed by a region of instructions that function as NOP equivalents [9]. By using an adjustable look-ahead size to match these NOP-like instructions, we can reliably generate consistent fingerprints for the static regions preceding the NOP-like instructions. In this method, using a look-ahead window of size *w* bytes, we search and see if all the bytes in this window can be found in a set of 76 breakmarks that have been identified. If they are, then we will generate a fingerprint using all the bytes that appear before this look-ahead window. After that, we begin a new search using a new window that begins 1 byte after the previously matched position. Figure 4 shows the operation of VPMB with three different window sizes: 5 bytes, 10 bytes, and 15 bytes. The same initial byte region is isolated in all three, producing one, consistent signature.

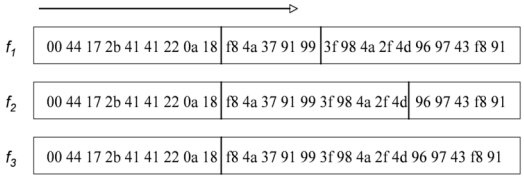

Figure 4. VPMB with three different window sizes

An appropriate choice of look-ahead size is required to reduce the false positives. Similar to the FPSW's dilemma, choosing a smaller size will increase false positives. But how small is too small? Through testing (as will be shown later), it has been determined that a size of 20 bytes reduces false positives to a minimum. Tested cases with values greater than 20 did not result in further reductions of false positive but incurred additional processing cost. Insight as to why 20 is the "magic" look-ahead size is the following - the probability of finding a grouping of NOP-like instructions in benign traffic drops considerably as the window size is increased. But in actual exploits, NOP regions tend to be larger than 20 bytes (so guessing an address back into the stack is a simpler process). Thus, 20 represents the point at which false positives drop to a minimum, and true positives don't require excess processing time.

4. Evaluation

To compare between the three approaches described in Section 3 and to evaluate the false positive/false negative rates of such a worm detection system, each of these algorithms has been implemented and applied to two one-hour traces that are extracted from a whole day's traffic trace that was kindly made available by the WAIL research group [10]. In this whole-day trace, traffic was observed from two /16 subnets (16K addresses) on two adjacent class B networks. Traffic from only one class C network contained within this trace is considered in this study. The total packet count for each 1-hour trace is 23,554 and 6,834 respectively. Each 1-hour trace is further divided into 5-minute intervals. Signature generation is performed on the traffic obtained in every 5-minute interval. To minimize the number of packets that the signature generation module needs to process, some simple filtering is performed on the trace: (i) only incoming packets destined for the target network are considered, (ii) only TCP packets with the PUSH flag set are taken for fingerprint generation. The methods, however, can be used for other attack packets (i.e. UDP-based attacks) as well. All data in each packet is considered for analysis (i.e., no static SNAPLEN is utilized).

4.1 FPSW

As previously mentioned, because of the potential large number of fingerprints that can be generated using the FPSW scheme, it is desirable to find ways to reduce the number of signatures to be retained for future intrusion detection purposes. To accomplish this, the simple IP address dispersion algorithm proposed

in [6] has been implemented. This algorithm is well suited for detecting rapidly spreading worms, by observing the frequency of distinct source and destination IP addresses of packets carrying a particular fingerprint. If a single fingerprint is sent from at least *n* distinct source IPs, and is destined to at least *n* distinct destination IPs, then, it is retained. A further trimming of the fingerprint pool is performed for each test by discarding fingerprints generated from data chunks with a high prevalence of NULL bytes ({00, 00, 00, 00, 00, 00, 00, 80}).

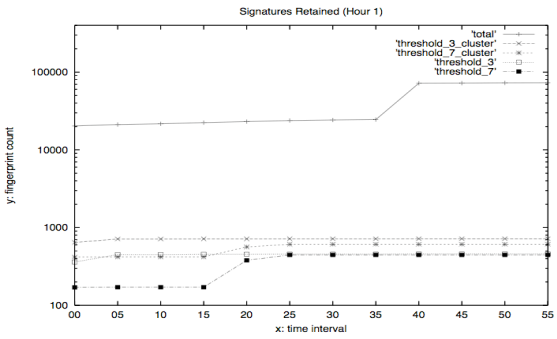

Figure 5. Signature counts for 1-hour trace

These fingerprints are far too general, and have little value for detecting malicious traffic. In conjunction with retaining the fewest possible signatures, we want to maximize the amount of useful signatures retained. To do so, the Levenshtein Edit Distance algorithm [11] has been chosen to find similar fingerprints (amongst those that have not been retained) to the ones that have been retained for each 5 minute interval

Figure 5 shows the result of applying the FPSB scheme against an hour long trace. Each graph interval (a five minute portion from the trace) increases cumulatively, with only new signatures being added to the total pool. Classification (using the algorithm described above) was performed at two different threshold intervals (*n*: $n = 3$ and $n = 5$. From the graph, we see that thousands of signatures are discarded at $n = 3$, while only a slight *decrease* occurs from $n = 3$ to $n = 7$. When the clustering algorithm (also described above) is applied at both classification thresholds, we see a slight *increase* in the signature pool, as expected. Via clustering, a total of 274 signatures are added to the signature pool, with about 23 signatures added per 5-minute interval. Thus, we are able to tune the signature pool accordingly, with the maximum amount of useful signatures retained.

From what has been shown thus far, it is clear that FPSW has the potential to generate a large number of signatures. Tables 1 and 2 provide us with insight into the dynamics of signature creation and retention. In both tables, the data has been condensed into six 10-minute windows. Table 1 presents the results for the formula Δpkts / Δnewflow, where Δpkts represents the additional (not total) packets from the new flows added to the total pool at each time interval, and Δnewflow represents the additional (not total) flows added at each time interval. In essence, this formula gives us an average number of packets added for each flow at each time interval. Overall, only a few packets are added at each time interval for both hours, with 2.66 packets added per flow on average for the first hour, and 3.05 packets for the second hour. Thus, the malicious traffic considered here does not produce a heavy flow of packets per connection. However, this is not a general trend that may be applied to all malicious traffic - denial of service attacks in particular may generate a very large packet pool per connection.

Time	Hour 1	Hour 2
00	2.49	3.18
10	2.65	2.65
20	2.30	2.86
30	2.73	3.65
40	3.46	2.79
50	2.33	3.17

Table 1. (Δpkts / Δnewflow)

Table 2 presents the results for the formula Δs / Δnewflow, where Δs represents the additional (not total) signatures added to the total pool at each time interval, and Δnewflow represents the additional (not total) flows added at each time interval. This formula gives us the average number of signatures added to the total signature pool per flow. All results for both classification intervals with and without clustering applied are included. Overall, only a small amount of new signatures is added by each flow.

Time	Hour 1 T = 3	Hour 1 T = 7	Hour 2 T = 3	Hour 2 T = 7	Hour 1 T = 3 cluster	Hour 1 T = 7 cluster	Hour 2 T = 3 cluster	Hour 2 T = 7 cluster
00	9	3.49	7.83	5.43	14.26	8.55	9.53	7.26
10	0.12	0	0	0.02	0.07	0	0	0.02
20	0.03	4.64	0.08	0.33	0.03	3.24	0.04	0.04
30	0	0	0.12	0	0	0	0	0
40	0.02	0	0.03	1.84	0.02	0	0.02	1.36
50	0	0	0	0	0	0	0	0

Table 2. (Δs / Δnewflow)

There are several interesting features present in this table. First, we notice the largest amount of new signatures added on average occurs at the first time interval, 00. This indicates a large amount of repetition in generated signatures at the time intervals that follow; which by time 50, no additional signatures have been added. Second, somewhat counterintuitive results are present at the threshold intervals where n = 7. From data presented in the earlier tables and graphs, we expect the numbers here to be lower than that of their n = 3 counterparts. In Table 2, we see several instances (Hour 1: 20, Hour 2: 40) where this is not the case. The explanation for such behavior is this - earlier flows where signatures were retained at n = 3 have been discarded by n = 7. However, these same signatures have been generated by retained flows at later time intervals. That is, the same packet data exists at multiple time locations, thus producing the same signatures.

Now that a pool of signatures has been generated, we must find out how relevant they are to detecting malicious traffic. Ideally, we would like the signatures generated early on to detect future instances of the same (or similar) threat. In practice, this is what occurs. The first number in each entry of Table 3 displays signatures from each time interval that are common to the initial retention period (time '00') for Welchia activity. The second number represents the total signature count. In Table 3, the greatest variance from the first time interval shown is 23 for hour 1, and a very large 275 for hour 2.

Investigation into why large variances may occur reveals that the rate of active protocol scanning by malicious software is the culprit. As opposed to stealth-type scanning performed by tools such as nmap (which may not even create a full TCP connection), active protocol scanning will probe a target port with data; if a desired response is received, exploitation will commence. FPSW will generate signatures for the scans, as well as the exploit attempts. Thus, a lull in scan activity will create variance in the count of similar signatures observed. The amount of additional signatures added to those already observed is low. For example, in Table 4, Hour 1, we see two intervals where the common signature count is lower than the initial interval: 05 and 15. In both these intervals, 274 total signatures are generated, with 85 of these signatures for each interval not found in the initial pool.

4.2 VPSW

VPSW, the breakmark identification scheme, faired poorly in testing. Because of the difficulty in isolating general use breakmarks ([5] makes no specific recommendations for breakmarks, only suggesting in note 6 that monitors choose them independently of one another), and the variance of data before the breakmarks, fingerprint retention rates were exceptionally poor. Limited success was achieved using a series of NOP instructions as a break-mark, with a few fingerprints from the Blaster worm retained. However, the modified version i.e., VPMB, works extremely well

Time	Hour 1 Threshold = 3	Hour 2 Threshold = 3
*00	212	298
05	(189, 297)	(297, 303)
10	(212, 297)	(297, 303)
15	(189, 297)	(23, 303)
20	(212, 297)	(297, 303)
25	(212, 297)	(192, 303)
30	(212, 297)	(298, 303)
35	(212, 297)	(292, 309)
40	(212, 297)	(297, 309)
45	(212, 297)	(297, 309)
50	(212, 297)	(284, 309)
55	(212, 297)	(298, 309)

Table 3. Signatures common to the initial retention period - Welchia

against polymorphic traffic. To test this method, multiple polymorphic versions of the Blaster worm, Welchia worm, and WebDAV Search exploit were created using the ADMmutate [18] kit. Another experiment was devised with two intents: (a) to see if consistent signatures are produced between different polymorphic versions of the same exploit, and (b) to see if the false positive rate will increase if more fingerprints are retained. In this experiment, two polymorphic worm packets of each type were injected into every 5-minute interval of the two hour-long traces and the VPMB scheme is used to see how many signatures are retained and how many worm packets of these 3 types are retained.

Table 4 shows the success and false-positive rate of the VPMB scheme. Using the 20-byte look-ahead window size, the VPMB scheme was able to identify all the worm packets that belong to these three types of malicious traffic. In testing, all six additional pieces of malicious traffic were detected, with only a single signature being generated for each distinct type. The

performance of VPMB is worse with a 10-byte or a 5-byte look-ahead window.

Time	signatures L-A = 5	false pos. L-A = 5	signatures L-A = 10	false pos. L-A = 10	signatures L-A = 20	false pos. L-A = 20
00	8	0.025	3	0.0	3	0.0
05	9	0.036	3	0.0	3	0.0
10	9	0.042	3	0.0	3	0.0
15	9	0.031	3	0.0	3	0.0
20	9	0.036	3	0.0	3	0.0
25	10	0.034	4	0.005	3	0.0
30	11	0.038	5	0.010	3	0.0
35	11	0.048	5	0.012	3	0.0
40	23	0.066	5	0.007	3	0.0
45	23	0.120	5	0.012	3	0.0
50	23	0.131	5	0.013	3	0.0
55	23	0.132	5	0.013	3	0.0

Table 5. Hour 1: VPMB, false positives - only three signatures are expected (signatures are cumulative

Additional packets are classified as "worm" packets because the generated signatures are not specific enough; for each time interval, only three signatures are expected. In Table 5, the false positive rate is the highest with a 5-byte look-ahead window, yet, even at this low look-ahead size, the worst interval, at '40', only adds 13 false signatures. With a 10-byte window, only one false signature is added at times '25' and '30'. Finally, with a 20-byte window, all false positives are eliminated.

5. Conclusion

In this paper we introduce three sliding-window based worm signature generation schemes: (a) method one, the fixed partition sliding window scheme (FPSW), (b) method two, the variable-length partition sliding window scheme (VPSW), and (c) method three, the variable-length partition with multiple breakmarks scheme (VPMB). Our evaluation indicates that the VPMB scheme is promising.

Several areas of further research for the sliding-window based methods have been identified over the course of this work. The classification scheme utilized in this paper is quite primitive, thus, more work must be made on finding new ways to retain useful fingerprints in an accurate and computationally efficient manner. The preliminary results from the clustering experiment are promising, and work will continue to pursue this approach in the future, implementing data mining and machine learning techniques. The ability of the VPMB scheme to detect polymorphic traffic is good, but the evolution of polymorphic techniques continues. Further toolkits and methods that can be used to obscure exploit code must continue to be evaluated, so the methods to combat them may advance.

Acknowledgements

We would like to thank Vinod Yegneswaran, Dr. Paul Barford, and the Wisconsin Advanced Internet Laboratory for their willingness to share several network traces that we have used in this paper.

6. References

[1] M. Roesch, Snort - lightweight intrusion detection for networks. LISA '99 - Proceedings of the 13th USENIX conference on System administation, Seattle Washington, 229-238, 1999.

[2] V. Paxson. Bro: a system for detecting network intruders in real-time. Computer Networks, Amsterdam, Netherlands, 31 (23-24): 2435-2463, 1999.

[3] M. Locasto, J. Parekh, S. Stolfo, A. Keromytis, T. Malkin, and V. Misra. Collaborative distributed intrusion detection. Tech Report CUCS-012-04, Department of Computer Science, Columbia University, 2004.

[4] L. Gao, J. Wu, S. Vangala, and K. Kwiat. An effective architecture and algorithm for detecting worms with various scan techniques. Proceedings of NDSS, 2004.

[5] H. Kim and B. Karp. Autograph: toward automated, distributed worm signature detection. Proceedings of the 13th Usenix Security Symposium, 2004.

[6] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting. Proceedings of the 6th USENIX Symposium on Operating Systems Design and Implementation, 2004.

[7] M. O. Rabin, Fingerprinting by Random Polynomials, Tech. Rep. TR-15-81, Center for Research in Computing Technology, Harvard University, 1981.

[8] A. Broder. Some applications of Rabin's fingerprinting method. In Renato Capocelli, Alfredo De Santis, and Ugo Vaccaro editors, Sequences II: Methods in Communications, Security, and Computer Science, pages 143--152. Springer-Verlag, 1993.

[9] K2. ADMmutate 0.8.4. Published online at `http://www.ktwo.ca/ADMmutate-0.8.4.tar.gz`. Last accessed on 6 Jan. 2006.

[10] R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of Internet background radiation. IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet

[11] V.I. Levenshtein, Binary codes capable of correcting spurious insertions and deletions of ones, Problems of Information Transmission, 1:8-17, 1965.