

Harnessing the power of P2P systems for fast attack signature validation

Spiros Antonatos*, Vu Quang Hieu†

**Institute of Computer Science,
Foundation for Research and Technology Hellas
antonat@ics.forth.gr*

*†Cryptography and Security Department
Institute for Infocomm Research, Singapore
{ghvu,kostas}@i2r.a-star.edu.sg*

Abstract—Attack signature validation plays a key role in intrusion detection and prevention technologies. Usually, when new attacks, particularly worms, appear, security software analyzes and generates signatures for these attacks. Since inaccurate signatures may block legitimate traffic that is similar to the attack traffic (false positives), security software is reluctant to deploy new signatures without extensive testing. The testing procedure, however, can be time consuming, resulting in significant delays (hours or even days) in signature dissemination. To alleviate this problem, in this paper, we propose a novel architecture based on P2P technology for fast content signature validation. The basic idea is to collect and store recent network traffic at peers participating in the system in advance and use it to validate new signatures. Since the amount of traffic that needs to be checked against is huge, we also propose a high-performance validation algorithm over stored traffic data. Experimental results show that our proposed system can validate candidate attack signatures and determine potential false positives rates in just a few seconds.

Keywords—indexing; signature validation; P2P defenses;

I. INTRODUCTION

Vulnerabilities in operating system and application software may cause severe consequences to computer systems. It is because attackers or virus programs can exploit these vulnerabilities to gain unauthorized access and damage the systems. As a result, to protect the systems, software vulnerabilities need to be frequently checked and when a new vulnerability is discovered, whether through detection or through code reviews and other “offline” mechanisms, it is typically followed by a distribution of software updates or patches. Ideally, the update process should be as fast as possible. Unfortunately, updates often require significant effort and time to develop, test and deploy. This creates a bottleneck in the reactive system defence lifecycle.

There are several approaches that target this bottleneck problem. The intrusion detection industry is developing intrusion prevention systems that can block suspicious traffic using the most reliable detection heuristics available. Similarly, techniques such as Microsoft’s Shield provide lightweight vulnerability-specific filters that can be implemented on the end-host by intercepting and analyzing incoming protocol messages. In both cases, the signatures or

filters to be distributed to users are reasonably small to be quickly pushed to a large number of sites, and much easier to compose than a permanent fully-blown update or patch. Since the inexact nature of new signatures introduces the risk of accidentally blocking legitimate traffic, the accuracy of these signatures need to be tested extensively. This testing procedure usually takes hours or even days because it needs to collect a large enough amount of traffic data to validate new signatures.

To speed up the process of signature validation, a solution is to collect and store recent network traffic in advance and use it to validate new signatures. This solution, however, has two main challenges: (1) how to collect and store a huge amount of network traffic efficiently and (2) how to retrieve stored data to perform fast signature validation. Since P2P systems are famous due to the capability of sharing resources so that giant data storages or systems can be built from sets of ordinary computers, we address the first challenge by building a distributed signature validation service on top of a P2P system. In our system, each participant peer regularly collects and stores its recent traffic data. When a new signature is submitted to the system for validation, the signature is tested against locally stored data at every peer. Since malicious peers can exist in the system and provide incorrect validation results, we also propose a method to alleviate the effects of malicious peers.

For the second challenge, inspired by work in the database community, to achieve fast signature validation, we trade space for time by creating indices for every piece of data. In this way, the system only needs to perform signature validation over data indices, which is fast. Since network data are binary while existing indexing techniques employed in databases only work for text data, we also propose a method that creates indices for binary data.

To sum up, our paper makes the following major contributions.

- We propose a distributed signature validation architecture on top of a P2P system where each peer in the system works as a signature validator using its recent traffic data.
- We introduce a method to create indices for binary

network data and use these indices to perform fast signature validation.

- We present a solution to avoid the effects of malicious peers in the system.

The rest of the paper is organized as follows. In Section II, we describe our proposed system architecture. In Section III, we present the algorithm to create indices for binary network data and perform fast signature validation. We discuss the solution to avoid the effects of malicious peers in Section IV. We show experimental study in Section V. Finally, we introduce related work in Section VI and conclude in Section VII.

II. SYSTEM ARCHITECTURE

In this section, we first discuss an overview of our system. After that, we present our system architecture in details and reasons why we choose it. Finally, we discuss challenges that need to be solved in order to use the proposed system.

A. Overview

We envision a distributed signature validation service where participants would be individuals or organizations that are willing to cooperate in order to lower false positives rates in their networks and receive benefits from early getting of new signatures. However, in exchange for these benefits, participants need to contribute to the system their recent network traffic data for signature validation, the storage to store traffic data, and the computational resource to validate new signatures. Due to privacy issues, each participant only captures, stores and performs signature validation on its own traffic data. In other words, each participant works as a independent signature validator. When a security company, particularly a signature development center, wants to validate a newly discovered attack signature, it could use our service by submitting the candidate signature to the system. This signature is then validated at all peers in the system. To validate a candidate attack signature, a peer needs to find any possible occurrence of the stored network traffic that matches the signature. By counting the number of matches the peer can derive a score for the signature. If the score is high, the new signature can be safely deployed. Otherwise, if the score is low, there is traffic with the same traffic characteristics as that of the attack. As a result, if the signature is deployed, a sort of Denial of Service (DoS) attack would probably take place. Based on the results obtained at peers, the system is able to tell whether the signature can be safely deployed.

B. System Architecture

In our system, we choose to build the proposed distributed signature validation service on top of a P2P system. In this way, each participant is actually a peer in the system. When a peer wants to join the system, it needs to register itself to the system and downloads a software to install. This

software consists of two main components. One component is in charge of collecting and storing recent local network traffic at the peer while the other is responsible for validating new signatures upon requested. As an example, Figure 1 shows a system with seven participants corresponding to seven signature validators. While it is possible to choose any type of distributed system to deploy our signature validation service, we choose P2P architecture because of there main reasons as follows.

- As stated before, false positives occur if the deployment of new signatures blocks legitimate traffic. Since the occurrences of false positives depend on traffic patterns, false positives rates may vary from network to network. As a result, it is desirable to validate new signatures over a variety of network traffic obtained from different locations/regions in the world. P2P architecture is particularly suitable to satisfy this desire.
- Intuitively, the more traffic our system uses to validate new signatures, the more confident the system would be that the given signature can be safely deployed. It means that to be able to guarantee the quality of signature validation results it is necessary to collect, store and process a huge amount of traffic data. Without employing P2P architecture, the cost to build a system for this purpose should be very high.
- Since the number of participants in the system could be potentially very large, other types of distributed systems cannot be employed efficiently.

C. Challenges

Even though the proposed system seems to work well, there are still a couple of challenges that need to be addressed. As mentioned above, the more traffic our system is storing, the more confident it would be that the signature under question can be deployed safely. Even though each peer in our system only needs to store and process its own traffic data, in order for the signature validation system to work satisfactory (i.e. to guarantee a low false positives rates), the peer needs to save at least the traffic of the last twenty four hours, which can still be large if the peer is actively sending and receiving data in the network. As a result, it may still take time to process this excessive amount of traffic to validate new signatures. To cope with this problem, we propose a solution where we trade space for time. In particular, we index every piece of the stored data so that the processing time later can be fast. We will present this solution in details in Section III.

The second challenge is how to guarantee the correctness of signature validation results. While we design the system open to everyone since we would like to have as many participants as possible, we need a solution to deal with malicious peers. Malicious peers can be attackers, who want to join the system to provide incorrect results for testing signatures. As a result, the system may produce a low score

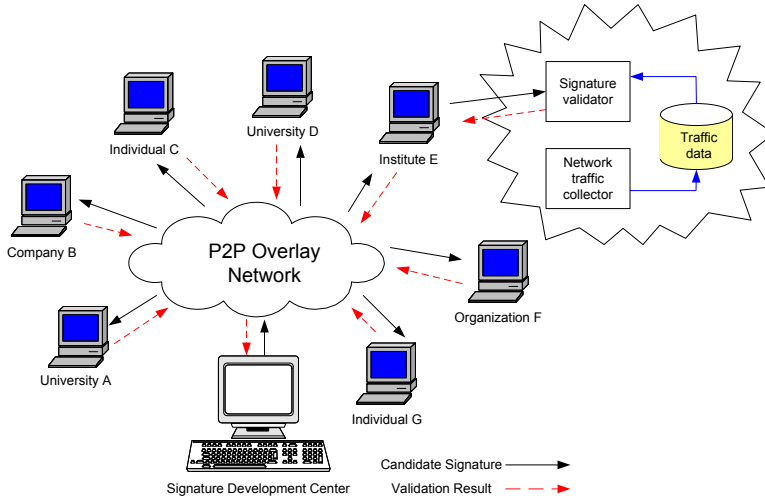


Figure 1. Distributed validation architecture design

for signatures that in reality can be safely deployed – or in fact must be urgently deployed. Our solution to this challenge is to keep track reputations of peers obtained in their previous transactions and use these recorded reputations to identify malicious peers. Furthermore, we organize peers in the system into two layers: the general layer includes all peers in the system while the trusted layer only consists of trusted peers. If the general layer cannot determine the correct result of a signature, the trusted layer is used to verify the result. Details of the solution will be discussed in Section IV.

III. SIGNATURE VALIDATION ALGORITHM

As mentioned previously, a major concern in our work is the time it takes to complete signature validation. Since it is desirable to perform a fast signature validation over a large amount of traffic data, we need to design an efficient algorithm. In our approach, we incorporate time-space trade-off techniques that are well understood in the context of databases. These techniques allow users to quickly scan for the existence of a key in a database. Since we expect from our algorithm a response time only in a few seconds, if our algorithm does not use any pre-processing of the captured traffic, only the time required to read the captured traffic from the storage device would be potentially great if the amount of data is big. As a consequence, we have to pre-process the captured traffic in order to minimize the data that our algorithm must read from the storage devices (hard disks). To be more specific, our algorithm has two phases, an “offline” phase where the captured traffic (trace) is processed and an index file is created, and an “online” phase where the algorithm uses the index file to validate the signature. We respectively present these two phases in subsequent parts of this section.

A. Offline phase

In the “offline” phase, the basic idea of the algorithm is to index every n -byte sequence appearing in the captured traffic. For every appearance of each sequence a 6-byte record is kept: 4 bytes for the packet number on which the sequence was found and 2 bytes for the position of the sequence inside the packet. As an example, Figure 2(a) shows indices of the 3-byte subsequences “aaa” and “aab”. The aim of our algorithm is to retrieve only the information stored on the index, eliminating the need to search on the real captured traffic. Thus, the size of information for each sequence must be as little as possible. By increasing n , the information stored for each sequence is reduced but the number of sequences we have to index increases. For example, choosing 1 as n , we only have 256 indices but each index is several megabytes (assuming a 1 GB input trace). Choosing 4 as n , each index contains a few records, but we have 2^{32} indices.

B. Online phase

Since the main idea to validate a signature is to count the number of occurrences of the signature pattern in the stored network traffic, the “online” phase is actually a search phase that performs exact matching based on the information stored in the indices. Initially, we retrieve the indices for the n -byte sub-sequences that form the pattern. We then correlate the retrieved information to find packets in which all sub-sequences are found and their positions are adjacent. Specifically, we first correlate the index of the first subsequence with the index of the second subsequence. Then, we check all 6-byte records to find those that have a common packet number. For instance, if a record of first index indicates that the first subsequence is found in packet A and packet A never appears on the records of the second

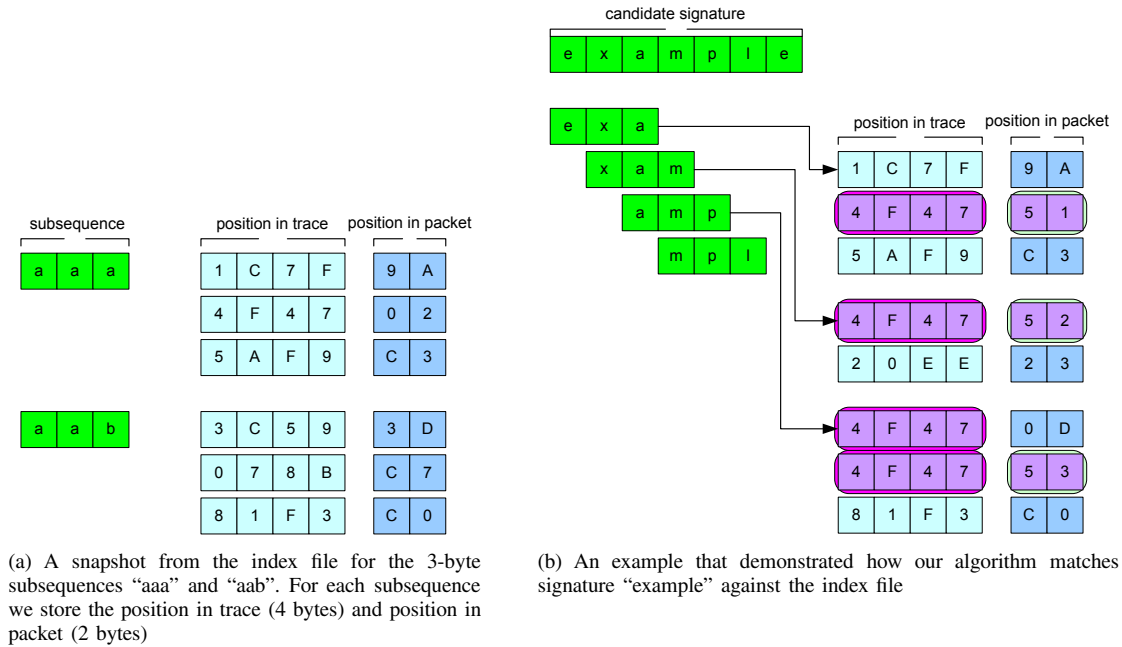


Figure 2. Outline of the indexing algorithm

index, then this record is dropped. For the records that have the same packet number, positions are checked. If in the first index there is a record saying "packet A position B", then we try to find if there is a record in second index that says "packet A position B+1". If such a record is found then the record of the second index is checked against the index of the third subsequence in order to locate a record "packet A position B+2" and so on. If the checks are successful up to the index of the last subsequence, then we have a match in packet A at position B. Note that we can apply the same technique to index header fields, such as IP addresses or TCP/UDP ports.

Figure 2(b) illustrates an example where a candidate signature is validated. Assume that we have $n = 3$, we need to search for the indexes of 3-byte sequences that form the signature, that is "exa", "xam", "amp", "mpl" and "ple". Note that to be neat, we only display indices of the three first subsequences in the figure. Our first step is to retrieve indices of "exa" subsequence. Since this subsequence is found in three packets 1C7F, 4F47 and 5AF9, these packets are candidates to include the signature. Our second step is to retrieve the indices of the next subsequence "xam". This subsequence is found in two packets 4F47 and 20EE. Since the subsequence is not found in packets 1C7F and 5AF9, we exclude these two packets from the packet candidate list. We further check packet 4F47 and since the subsequence appears in the adjacent position (52) in relation to first subsequence, packet 4F47 is still kept in the candidate list. As a third step, we retrieve indices for subsequence "amp". This subsequence is found in packet 4F47 and in

an adjacent position in relation to the previous subsequence so this packet continues to be a candidate. This process continues until the last subsequence and if all subsequences are contained in packet 4F47 and in adjacent positions, we can say for sure that this packet contains the signature "example". Furthermore, we are able to answer the position of the signature inside the packet, that is "example" is found at packet X in position Y. This is useful as many signatures contain information about depth and offsets of the string they search for.

Our approach needs 6 times more space than the original trace in worst case. However, during the index creation we perform several optimizations. If we have a sequence that contains the same character, for example ten consecutive appearances of character A, then we do not store 10 indices but a single one that denotes we see character A at a given offset and for 10 times. This optimization has decreased our storage needs to a factor of 5.2. A second optimization that improves our speed is a in-memory bitmap that keeps track if a subsequence has stored information or not. In cases where we choose to index 4-byte sequences, many subsequences do not appear in the traffic trace. The in-memory bitmap provides a fast way to find out if a subsequence has stored indices or not; if the signature contains a subsequence that has no stored indices, then we know for sure that this signature will never match. Before we start the matching algorithm, we check that all subsequences that form the signature have stored indices. If not, the searching algorithm as described earlier is invoked.

IV. DEALING WITH MALICIOUS PEERS

As mentioned above, a problem with the openness of our proposed system is that malicious peers or attackers may attempt to join the system and provide incorrect signature validation results. Since a signature validation result can be either “pass” or “not pass”, there are two cases of an incorrect result.

- The testing signature is incorrect, and hence it should not be passed. In this case, if a malicious peer provides incorrect result, i.e., it votes to pass the signature, when the signature is deployed, the malicious peer is the one that is affected by the deployed signature. As analyzed before, a sort of DoS attack would take place at the malicious peer because the signature pattern exists in its network traffic. Furthermore, even though the malicious peer may sacrifice itself to accept such an effect in order to also harm other peers, the signature will not be passed with high probability since a majority of other peers will vote against the signature.
- The testing signature is a correct signal of an attack, and hence it should be deployed. In this case, if a malicious peer provides incorrect result, i.e., it votes against the signature, the system may not pass the signature. This is the case we should worry about. As a result, for the rest of this section, we only discuss a solution to deal with this case.

A. Identifying Malicious Peers

To identify malicious peers, we employ a typical approach that is often used to evaluate the trustfulness of a peer in P2P systems. In this approach, the trustfulness of a peer is evaluated through previous transactions the peer has done. In our system, the trustfulness of a peer is evaluated based on the correctness of results the peer provides in previous transactions. In particular, when a peer first joins the system, it is assigned a predefined trust score. Every time the peer provides a correct result, its trust score is increased. On the other hand, each time the peer gives an incorrect result, its trust score is either reduced or kept unchanged depending on the concrete situations we will discuss in the next part. In this way, looking at the trust score of a peer, we can determine how much we trust that peer. If the trust score is high, we can trust the peer while if the trust score is low, we should not trust the peer. Knowing the trust score of peers, we can put a weight on their results, i.e, results from peers having high trust scores should have more influence to the final result than results from peers having low trust scores. The remaining task is how to determine if a result is correct or not. We discuss in the next part of the section.

B. Verifying Signature Testing Results

Intuitively, the correctness of the result can be determined by the result of a majority obtained at peers. Assume that the acceptance false positives rates is K percents (a predefined

value). In this case, let S_1 be the number of peers that pass the signature to be deployed and S_2 be the number of peers that do not pass the signature, M be the number of malicious peers that can exist in the system and let $F = \frac{S_1}{S_1+S_2}$, we consider three cases of F .

- $F < K$: since F falls into the acceptance rate of false positives, we simple consider that the signature is validated. In this case, we increase the trust score of peers providing the correct and decrease the trust of peers providing incorrect results.
- $F \geq K + M$: since after eliminating M results, the number of peers that vote against the signature is still greater than or equal to K , the correct result is that the signature is not passed. In this case, we increase the trust score of peers providing the same result to the final result and keep the trust score unchanged for other peers.
- $K \leq F < K + M$: since we cannot determine the correct result easily, our solution is to employ a second layer of trusted peers to verify the result. Trusted peers are participants that are well-known organizations. For example, all peers in PlanetLab (www.planet-lab.org) can be trusted because they are contributed from universities in the world and are put under strict controls. These participant cannot be malicious peers and hence we can totally trust their returned results. It means that the final result will be the result obtained from these trusted participants. In this case, we increase the trust score of peers providing the correct result while decrease the trust scores of peers supplying incorrect results.

C. Dealing with Attacks on the System

Since the system is proposed to provide fast attack signature validation, and hence a fast reaction against attackers, the system itself can suffer attacks from attackers. A typical attack can be a Sybil attack, where an attacker (or a group of attackers) owns a large number of peers in the system and uses them to gain influence. Alternatively, knowing that high score trust peers have more effect on the evaluation process, attackers can try to attack and control these peers to provide incorrect results. Our solution to this problem is that instead of asking all peers in the system involved in the evaluation process, we only choose randomly a subset of peers to evaluate signatures. Assume that we have a large number of participants in the system, a subset of them is till enough to evaluate new signatures. In this way, since the peers involved the evaluation process is randomly selected, malicious peers cannot easily cooperate to gain influence while attackers cannot know which peer is selected to attack. As a result, the system remains strong against attacks. Furthermore, to resist a whitewashing attack, where an attacker repeatedly re-joins the system to avoid the previous bad reputation, we

suggest that the higher the trust score of a peer is, the higher the probability it is selected in the evaluation process.

V. EXPERIMENTAL STUDY

To evaluate the performance of our proposed signature validation service, we implemented a prototype of the system and tested the prototype on signatures found on Snort [1], a popular intrusion detection system. We tested these signatures against the two real network traces: FORTH.webtrace and Nlanr.MRA. Both traces contain 3GB of data. While FORTH.webtrace is a trace captured during a portal mirroring, Nlanr.MRA is a trace with random payload. Since we only have two real network traces, we just tested the prototype on two computers, each processed a network trace. Note that even though we used only two computers to test, the scalability of the system is still guaranteed since we employed P2P architecture for our service.

A. Overhead Cost

Since the cost of managing trust scores of peers is ineligible compared to the cost of processing network traces to create indices for signature validation, we only focus to evaluate this overhead cost. In particular, we evaluated the time to process network traces to create indices and the storage required to hold indices. In terms of processing time, our method takes less than an hour to generate indices for these experimental network traces. In terms of storage cost, our method requires an amount of storage 10 times greater than the size of the network traces (2 bytes for packet position, 4 bytes for packet number, and 4 bytes for next pointer) plus a fixed 64MB pointer table (3-byte sequences) or 16GB pointer table (4-byte sequences).

B. Signature Validation Performance

The results for 3- and 4-byte sequences are summarized in Figure 3(a). For almost 95% patterns we tested, our algorithm achieves sub-second search time. Our preliminary results also indicate that hotspots presented in the rest 5% of patterns are one to two orders of magnitude more effective than traditional linear searches. The dominant cost of our approach is the size of the indices we need to retrieve. The size of each index for two traces is presented in Figure 3(b). For almost 95% of sequences, we need to retrieve up to 1 Kbyte, although the maximum value reaches 16 Mbytes due to some popular sequences like consecutive zeros found in JPEG images. In the ideal case, this of random payload, each index is 500 to 900 bytes long. As the data we have to retrieve from disk are only a few kilobytes for the 3 GBytes trace, we can expect that time for searching on Terabyte traces is also near one second. Either fetching a few Kilobytes or a few Megabytes (e.g. less than 20MB) from a local hard disk requires almost the same time.

For comparison purposes we used Snort to validate some of its own signatures. The result shows that without considering the time to collect Snort required around 80 seconds

to validate a signature on a 3 Gbytes trace. Most of the time is spent on reading the packet trace from the disk, while the user time is nearly 3 seconds. Doing the same validation with our algorithm takes around 1 second for 80% of the possible patterns.

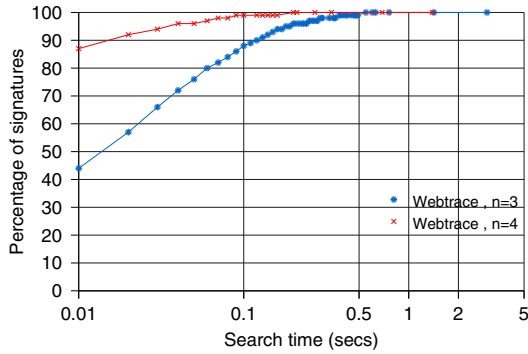
VI. RELATED WORK

Current practice suggests that NIDS signatures for new worms are usually available several hours or days after the initial worm outbreak, due to the manual signature generation process. This implies that, given the propagation speeds of worms so far, signatures are available after the worm has infected the majority of its victims. In such timescales, and with theoretic results showing that well-prepared worms can infect the majority of the vulnerable population in less than 10 minutes [2], it becomes clear that human-mediated containment methods cannot provide an effective defense against fast spreading worms [3].

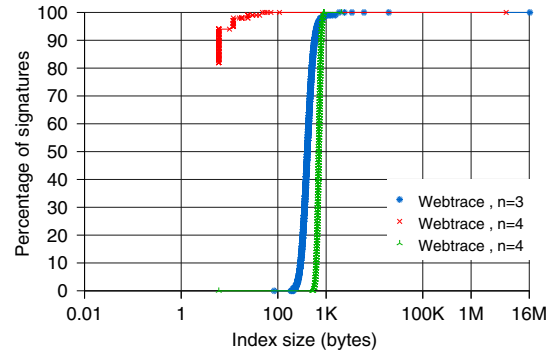
Two of the earliest network level worm detection and signature generation systems, Earlybird [4] and Autograph [5], rely on the identification of invariant content that is prevalent among multiple worm instances, a technique called *content sifting*. When sufficient worm instances have been identified, these techniques generate NIDS signatures by finding the longest contiguous byte sequence that is present in all instances.

EarlyBird [4] is based on two key behavioral characteristics of Internet worms: the prevalence of invariant content among different worm instances, and the dispersion of the source and destination addresses of infected hosts. Earlybird operates on network packets captured passively at a single monitoring point, such as the DMZ network of an organization. For each packet, digests of all 40-byte substrings are computed incrementally using Rabin fingerprints. The most prevalent digests among all observed packets to the same destination port are identified using space-efficient multi-stage filters. The intuition behind grouping by port number is that repetitive worm traffic always goes to the same destination service, while other repetitive content such as popular web pages or peer to peer traffic often goes to ports randomly chosen for each transfer. Furthermore, in order to ascertain that packets with recurring contents belong to a current epidemic, the number of distinct source and destination IP addresses seen in these packets should reach a certain threshold before raising an alert.

Similarly to Earlybird, Autograph [5] is an automated signature generation system for previously unknown worms. In contrast to Earlybird, which inspects all monitored traffic, Autograph uses a first stage portscanning-based flow classifier for identifying potential malicious connections. Autograph performs TCP stream reassembly for the inbound part of TCP connections, which reconstructs the client-to-server stream from the incoming packet payloads. The resulting suspicious reassembled streams, grouped by destination port



(a) of search time for $n=3$ and $n=4$ for a trace containing web traffic



(b) of sizes for indices for a trace containing web traffic and a trace with random payload

Figure 3. Cumulative distribution functions

number, are fed to the second signature generation stage. The signature generation algorithm searches for repeated non-overlapping byte sequences across all suspicious streams using Rabin fingerprints.

Akritis et al. present a worm detection and signature generation approach based on content sifting [6], similar to Earlybird and Autograph. The proposed technique operates on reassembled TCP streams, but explicitly discards traffic sent from servers to a clients and processes only client requests which may carry a worm payload. Prevalent substrings are found using Rabin fingerprints and value-based sampling, which reduces significantly the processing overhead. Other optimizations over previous approaches include using substrings of 150–250 bytes instead of the 40-byte strings used by Earlybird, which significantly reduces false positives, and giving a higher weight to substrings found in the first few kilobytes of a new stream, which effectively discards repeated control messages of peer-to-peer protocols that appear like worm attacks, but are sent over already established connections that have already transferred a significant amount of data.

Polygraph [7] generates signatures for polymorphic worms by identifying common invariants, such as return addresses, protocol framing, and poor obfuscation, which the authors argue that are present among different polymorphic worm instances. The key difference to previous automated signature generation approaches is that Polygraph looks for *multiple disjoint* byte sequences, which may be of very small size, instead of single contiguous byte sequences. The authors explore the effectiveness of different classes of signatures. Specifically, among longest substring, best substring, conjunction, token subsequence, and Bayes signatures, token subsequence signatures are the most effective, with the lowest false positive rate. Token subsequence signatures consist of multiple disjoint substrings and can be expressed with regular expressions. Bayes signatures are similar to

token subsequence signatures, but allow for probabilistic rather than exact matching, by assigning an occurrence probability to each token. Similarly to Polygraph, Hamsa [8] is a network-based automated signature generation system for polymorphic worms. Using greedy algorithms, Hamsa achieves significant improvements in speed, noise tolerance, and attack resilience over Polygraph.

Honeycomb [9] is probably the first automated system for the extraction of attack signatures from network traffic. Honeycomb has been implemented as an extension to the honeyd [10] low interaction honeypot, which groups connections to the same destination port, and attempts to extract patterns from the exchanged messages. Pattern detection is performed by applying the longest common subsequence (LCS) algorithm to the messages of the different connections in the same group, in two different ways. Given a sequence of messages, *horizontal detection* applies the LCS algorithm to the i th messages of all connections with the same destination number. In contrast, *vertical detection* first concatenates the incoming messages of individual connections, and then applies the LCS algorithm to the different concatenated streams.

Nemean [11] is a system for automated signature generation from honeynet traces, aiming to reduce the rate of false positives by creating semantics-aware signatures. The architecture consists of two components: the data abstraction component (DAC) and the signature generation component (SGC). The DAC takes as input a honeynet trace and first normalizes the packets for disambiguating obfuscations at the network, transport, and application layers (for HTTP and NetBIOS/SMB protocols). Then, it groups packets to flows, flows to connections (request-response message collections), and connections to sessions, which are defined as sequences of connections between the same pair of hosts. Normalized sessions are finally transformed into XML-encoded semi-structured session trees, suitable for input to the clustering

module. In the SGC, a clustering module groups connections and sessions with similar attack profiles according to a similarity metric. Then, the automata learning module constructs an attack signature from a cluster of sessions using a machine-learning algorithm. Due to the hierarchical nature of the session data, Nemean produces separate signatures for connections and sessions, appropriate for usage with the Bro NIDS [12].

Wang *et al.* [13] propose applying content-sifting techniques within clusters of traffic, as created with header-based multi-dimensional flow clustering. In many cases, this will improve the purity of signature pools. Wang and Stolfo [14] use byte distributions (1-gram frequencies) in payloads to identify traffic that deviates from normal (based on training).

VII. CONCLUSION

Attack signature validation based on P2P architecture, as proposed in this paper, has the potential of enabling security vendors to very quickly get feedback about the quality of a candidate signature. As a result, it becomes possible to reduce the time between initial discovery of an attack and the deployment of the appropriate response to intrusion detection and prevention systems. Furthermore, the high-performance algorithm outlined in this paper enables the required checks for the validation of the candidate signatures to be performed rapidly on large datasets, in order to reduce the statistical probability of false positives. We have presented an architecture, the core algorithm, and have validated the approach experimentally using real-world data. Our results indicate that it is possible to test a content signature against a 3GB trace in just a few seconds. The normal execution time of the Snort IDS without considering the time for collecting data for the same trace is around 80 seconds. This is a significant improvement that brings testing time in line with worst-case malware propagation times. While much work remains to be done in closing the loop in automated, reactive defenses, we believe our work to be a significant contribution in this direction.

ACKNOWLEDGMENTS

This paper is part of the 03ED440 research project, implemented within the framework of the “Reinforcement Programme of Human Research Manpower” (PENED) and co-financed by National and Community Funds (25% from the Greek Ministry of Development-General Secretariat of Research and Technology and 75% from E.U.-European Social Fund). Spiros Antonatos is also with University of Crete.

REFERENCES

- [1] M. Roesch, “Snort: Lightweight intrusion detection for networks,” in *Proceedings of USENIX LISA*, November 1999, (software available from <http://www.snort.org/>).

- [2] S. Staniford, D. Moore, V. Paxson, and N. Weaver, “The top speed of flash worms,” in *Proceedings of the Second ACM Workshop on Rapid Malcode (WORM)*, 2004, pp. 33–42.
- [3] S. Staniford, V. Paxson, and N. Weaver, “How to Own the Internet in Your Spare Time,” in *Proceedings of the 11th USENIX Security Symposium*, Aug. 2002.
- [4] S. Singh, C. Estan, G. Varghese, and S. Savage, “Automated worm fingerprinting,” in *Proceedings of the 6th Symposium on Operating Systems Design & Implementation (OSDI)*, Dec. 2004.
- [5] H.-A. Kim and B. Karp, “Autograph: Toward automated, distributed worm signature detection,” in *Proceedings of the 13th USENIX Security Symposium*, 2004, pp. 271–286.
- [6] P. Akritidis and E. P. Markatos, “Efficient content-based detection of zero-day worms,” in *Proceedings of the IEEE International Conference on Communications (ICC)*, May 2005.
- [7] J. Newsome, B. Karp, and D. Song, “Polygraph: Automatically Generating Signatures for Polymorphic Worms,” in *Proceedings of the IEEE Security & Privacy Symposium*, May 2005, pp. 226–241.
- [8] Z. Li, M. Sanghi, Y. Chen, M.-Y. Kao, and B. Chavez, “Hamsa: Fast signature generation for zero-day polymorphic worms with provable attack resilience,” in *Proceedings of the IEEE Symposium on Security & Privacy*, 2006, pp. 32–47.
- [9] C. Kreibich and J. Crowcroft, “Honeycomb – creating intrusion detection signatures using honeypots,” in *Proceedings of the Second Workshop on Hot Topics in Networks (HotNets-II)*, Nov. 2003.
- [10] N. Provos, “A virtual honeypot framework,” in *Proceedings of the 13th USENIX Security Symposium*, Aug. 2004, pp. 1–14.
- [11] V. Yegneswaran, J. T. Giffin, P. Barford, and S. Jha, “An architecture for generating semantics-aware signatures,” in *Proceedings of the 14th USENIX Security Symposium*, August 2005.
- [12] V. Paxson, “Bro: A system for detecting network intruders in real-time,” in *Proceedings of the 7th USENIX Security Symposium*, January 1998.
- [13] J. Wang, I. Hamadeh, G. Kesidis, and D. J. Miller, “Polymorphic Worm Detection and Defense: System Design, Experimental Methodology, and Data Resources,” in *Proceedings of the 1st Workshop on Large-Scale Attack Defence (LSAD)*, September 2006, pp. 169–176.
- [14] K. Wang and S. J. Stolfo, “Anomalous Payload-based Network Intrusion Detection,” in *Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID)*, September 2004, pp. 201–222.