ANALYSIS OF AUTOMATED NETWORK ATTACK SIGNATURE GENERATION MODELS

kapoorshray@gmail.com

Contents

1. Overview

2. Signature Generation without Attack Detection

    2.1 Polygraph
        2.1.1 Concept
        2.1.2 Technical Details
        2.1.3 Parameters
        2.1.4 Final Word- Conclusion

    2.2 Honeycomb
        2.2.1 Concept
        2.2.2 Technical Details
        2.2.3 Parameters
        2.2.4 Final Word- Conclusion

    2.3 EarlyBird
        2.3.1 Concept
        2.3.2 Technical Details
        2.3.3 Parameters
        2.3.4 Final Word- Conclusion

    2.4 Nemean
        2.4.1 Concept
        2.4.2 Technical Details
        2.4.3 Parameters
        2.4.4 Final Word- Conclusion

3. Signature Generation with Attack Detection

    3.1 Autograph
        3.1.1 Concept
        3.1.2 Technical Details
        3.1.3 Parameters
        3.1.4 Final Word- Conclusion

Content

3.2 P.A.D.S
- 3.2.1 Concept
- 3.2.2 Technical Details
- 3.2.3 Parameters
- 3.2.4 Final Word- Conclusion

3.3 PAYL
- 3.3.1 Concept
- 3.3.2 Technical Details
- 3.3.3 Parameters
- 3.3.4 Final Word- Conclusion

4. Conclusion

1. Overview

The aim of this research is to analyze various automated signature generation approaches , catering to generation of efficient signatures for polymorphic worms and networks attacks . Automated signature generation algorithms help in generating signatures for intrusion detection systems to combat network Zero-day attacks on fly, whose signatures are not available. Polymorphic worms and network attacks specifically change their payloads frequently to bypass IDSes signature detection. To detect these types of attacks either heuristic detection should be incorporated with rule-based detection or some algorithm needs to be implemented to generate signatures for polymorphic attacks . This deliverable provides an assesment of the feasibility for the following different approaches and their practical implementation , to generate efficient signatures, on high traffic networks.

Signatures Generation without detecting attacks

1. Honeycomb [1]
2. Polygraph [2]
3. Earybird [7]
4. Nemean, etc.

The above category focuses on generating signatures taking malicious as well as innocuous network data, without knowing an attack. Network sensors are placed on central nodes or routers which monitors every packet passing through it.

Signature Generation with attack detection

1. Autograph [5]
2. PADS
3. PAYL [4] [1]

Signature generation with attack generation , monitors only the malicious traffic filtered using honeypots or IDS itself and passed to the systems which will analyze the attack against knowledge based on previous flows.

Every algorithm derives a longest common string unique for every flow of worm (or attack) , and generates either a contiguous or a token-subsequence signature.

Our study is based on describing concept , technical details and parameters which analyze each approach . We have arrived with following parameters which are almost common with all the above techniques :-

False Positives

False positives are the erroneous alarms which an IDS pops-up, with normal communication identifying as abnormal. False positives are major concern; indeed this parameter benchmarks any system for its efficacy.

Detection Rate

Detection rate in terms of Signature generation algorithms, measures the efficiency of signatures in approximating an attack. High detection rate and low false positives are major requirements for these systems.

Time

Time plays an important role, as knowledge based systems are not time critical , signature generation requires sufficient knowledge and to be effective enough. There is a need for time criticality in signature generation approaches , so as to generate a rule for the attack before its impact. This acts as an important factor for zero-day attacks , because sufficient knowledge for those attacks is not available at hand.

Testing environments

To effectively benchmark an approach , not only its results are important but also the testing environment in which those results are generated. Testing environments should be close to real practical environments in which these systems will be placed, or otherwise test results are of no worth practically.

Types of attacks

Some approaches are specific to generating rules for polymorphic worms , while other focus on general network attacks. Testing for polymorphic worms needs more than one sample of that morphed worm , while for network attacks many such samples may not be available. So testing environment and types of attacks are catered to the practicality of approach and its motivation.

Flow pool

As we have categorized our study in two groups i.e. signature generation with attack detection and without attack detection, flow pool for testing for these two groups , need to contain enough only malicious data and malicious + benign data respectively. If an approach alls in the later category their should be enough noise to be injected so as to claim scalability of an algorithm.

Types of signature generated

Effectiveness of signature generation algorithm heavily depends on the type of signature generated. Single substring contiguous signatures raises low false positives but have high false negatives. They can oversee an attack due to strict and specific rule. Conjunction signatures on the other hand contains ordered set of tokens , less restrained than single contiguous signature . Score based signatures and unordered

token-subsequence signatures are considered highly effective for high detection rate and low false negatives.

Some of the above parameters are not addressed by some approaches which will be covered in next sections in detail. Section 2 and Section 3 describes and analyzes techniques for Signature generation without attack detection and with attack detection respectively . Finally Section 4 concludes the paper.

2. Signature Generation Without Attack Detection

2.1 Polygraph

2.1.1 Concept

Polygraph is a signature generation approach which uses innocuous as well as malicious flow pool for analysis of network traffic. This method focuses on polymorphic worms and extends its concept to polymorphic malicious network traffic in an attempt to bypass static signatures . The approach is based on the concept that their occurs some invariant byte code in any exploit which makes the exploit work. The absence of this invariant byte code will render the worm unfruitful. In addition to invariant bytes , there are some wildcard bytes which can be inserted without interfering the execution of worm , and code bytes which are the real instructions of that worm. Code bytes may be changed, for example a polymorphic worm can decrypt those code bytes with some decryption algorithm to be undetected , while the signature of decryption routine cannot be obfuscated. Overall studies says , there must exist some invariant bytes either bytes of decryption routine or execution instructions itself, which will help in framing signatures. Designers of Polygraph analyzed some polymorphic attacks to prove this concept.

Apache multiple-host-header vulnerability

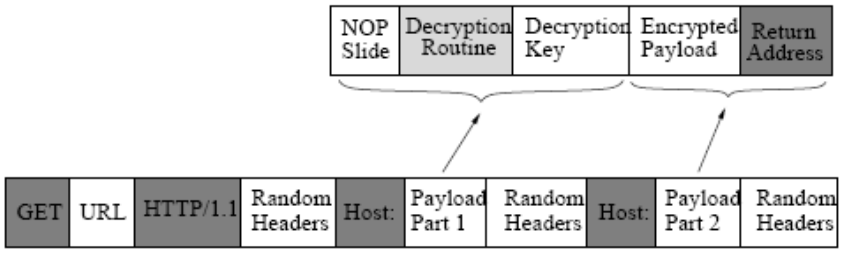

Fig. 1

Above is a typical header information for the attack. The payload contains several invariant protocol framing strings: “GET”, “HTTP/1.1”, and “Host:” twice. The second Host field also contains an invariant value used to overwrite the return address.

Code Red Worm

Code red worm overflows the buffer when converting ASCII to Unicode format. The exploit must be a GET request to an .”ida” file , so payload of this worm must contain GET and “.ida” as invariant fields.

Above examples proves the worth of concept , there were more such examples as discussed in the paper which determines that signature can be framed on tokens of this invariant bytes.

Architecture of Polygraph Monitor

Polygraph monitor can be placed on a router or any node in a network which can monitor all inbound and outbound traffic.

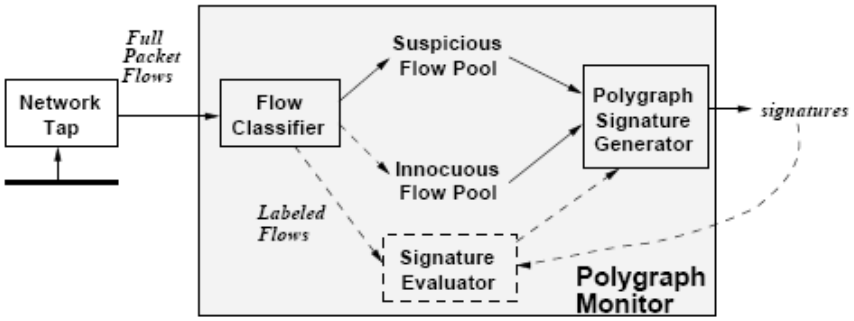

Fig. 2.

Flow Classifier as seen in the figure, classifies flow in either suspicious pool or innocuous pool by reassembling the packets and storing information of flow in IP:Port format. Flow classifier uses heuristics to classify flow, some heuristics are , monitoring the execution of server to detect exploits at runtime and mapping those to packet payloads , or to use honeypot for filtering malicious traffic. These techniques are not useful in isolating one type of malicious flow with other, but however they can be used in flow classifier to at least filter the noise. Design of flow classifier is not framed by polygraph designers , but they have used some pre-defined methods for classifying.

Malicious flow pool and benign traffic , assumed to be filtered by Flow Classifier, enters Polygraph Signature Generator .Design goal of PSG is to generate efficient invariant tokens to classify an attack.

2.1.2 Technical Details of PSG

Aim of PSG is to generate either one signature or a set of signatures matching a specific malicious flow. Prior to Signature generation , we need to extract distinct tokens on which signatures will be framed. Approach for token extraction consists of extracting of distinct strings of minimal length ‘L’ which occurs in at least *K* out of *n* samples. Strings which are substrings of token itself

are disregarded. For example a string "GET" will occur in most of HTTP packets, substring "ET" will thus occur in all those packets. We need to discard those substrings which are not distinct unless and until they occur in *K* out of *n* samples not as a substring of "GET". After extracting tokens a single signature which matches all flows in suspicious flow pool can be generated. This signature will consist of all set of tokens extracted above. There are many types of signatures which can be generated using polygraph, for example token-subsequence signatures, Bayes signatures and conjunction signatures. By far token subsequence signature is evaluated to be most efficient against any other signature. . To generate a token-subsequence signature, we want to find an ordered sequence of tokens that is present in every sample in the suspicious pool. Token sub-sequence approach requires clustering of flow pool Similar malicious flows are clustered hierarchically, and a signature is generated for each cluster. Hierarchical clustering starts with number of flows (flows means all packets having same IP:Port information) and generating signature for each flow. Consecutive steps involves merging the clusters and generating an efficient signature for the merged cluster . If no signature can be generated than those clusters are different and have no common signature. These merged signatures can be tested against the innocuous flow pool to estimate the false positive and false negative rate for the merged signature. Merging stops if the false positive rate is unacceptably high for the merged clusters.

Token sub-sequence algorithm

To generate a token-subsequence signature, we want to find an ordered sequence of tokens that is present in every sample in the suspicious pool. A subsequence of common strings in two samples can be found by analyzing payload and picking ordered set of tokens which are same in both samples, may not be consecutive. The problem of finding longest common substring in two samples is a string alignment problem. For example in the strings "xxonexxxtwox" and "oneyyyyytwoyy", the longest common subsequence is "onetwo". "onetwo" does not occur with same difference and depth but ordering is same, so this can used as a longest substring. An alignment is assigned a score by adding 1 for each character that is aligned with a matching character, and subtracting a gap penalty Wg for each maximal sequence of spaces and/or non-matching characters. That is, there is a gap for every ".*" in the resulting signature. However, we do not count the first and the last ".*", which are always present. In our experiments, we set Wg to 0.8. For example we have a common substring ".*o.*n.*e.*z.*", which means occurrence of "o" than some characters (.*) and so on. We have 4 characters so and 3 gaps, so score for this signature is 4-3*.8= 1.6 . If the score is good enough we prefer that signature else will analyze some other common longest substring.

Thus with hierarchical clustering and token-subsequence algorithm we will be able to find signatures for polymorphic worms with the assumption that we have enough samples of that worm for our classification.

2.1.3 Parameters

Types of signature generated

Polygraph can generate Bayes signatures, token-subsequence signatures and conjuction signatures. As discussed above conjuction signatures focus on subsequence string of unordered tokens present in every sample of suspicious pool for the same worm sample. Conjuction signatures give rise to false negatives , because of their behavior of unordered set. Strength of polygraph as argued by its designers is its ability to generate ordered token-subsequence signatures , which are more specific than either of the other approach.

Types of attacks

Polygraph assumes more than one sample of worm data to be available for analysis , which may not be efficient in real-time systems. As the design is catered to generate signatures for polymorphic attacks or worms, this approach should require multiple and as many as samples possible (50+) for finding a token-subsequence signature perfectly matching multiple worm samples. This approach is not well suited for network Zero-day attacks, for which multiple sample data is not available. Moreover the attacks discussed above considers some invariant payload for polymorphic worms which is not a concrete assumption. If we introduced extra fixed tokens and keep removing them with time , the system will keep on generating new signatures for the same worm. Bayes signature is the solution for the above problem, it focuses on assigning a score for each token and maintaining a threshold for highest false positive rate . However Bayes signature depend on probability for matching a signature given a threshold, so it is a less rigid approach than conjunction or token-subsequence approach.

Flow Pool

Polygraph monitors every flow passing through its sensors , malicious and benign. Design of Flow classifier is not specified and not within the scope of polygraph, however noise has a major impact in malicious flow pool which can trick the system in generating signatures giving false positives. So there is no question of training data to be noisy, it is assumed to be noisy containing both useful and useless data.

Testing environment

In all the experiments, the token-extraction threshold was set to k = 3 , the minimum token length a = 2, and the minimum cluster size to be 3. 5 independent trials for each experiment were conducted All experiments were run on desktop machines with 1.4 GHz Intel Pentium R III processors, running Linux kernel 2.4.20 . Polymorphic engine was used to generate polymorphic payloads for Apache-Knacker exploit and BIND-TSIG exploit. Network traces

for 5 days from Intel Research was used to test the signature for false positive rate. Numbers of worms needed were greater than 2, which is the constraint for the system. Tests were not conducted for different types of worms (only for the ones which were analyzed during the framing part) not even for zero-day polymorphic attacks.

Results and Detection Rate

Results of above experiment showed polygraph to be more efficient for HTTP and DNS based attacks. If the sample size is less than 2 , than 0% of the time worm signatures were efficient. While for samples greater than 2 , signatures were generated efficient for 100% of the time for all experiments.

Figure below shows different types of signature and their false positives rates as generated , for the Apache-Knacker exploit .

Class	False +	False −	Signature
Longest Substring	92.5%	0%	HTTP/1.1\r\n
Best Substring	.008%	0%	\xFF\xBF
Conjunction	.0024%	0%	'GET ', ' HTTP/1.1\r\n', ': ', '\r\nHost: ', '\r\n', ': ', '\r\nHost: ', '\xFF\xBF', '\r\n'
Token Subsequence	.0008%	0%	GET .* HTTP/1.1\r\n.*: .* \r\nHost: .* \r\n.*: .*\r\nHost: .*\xFF\xBF.*\r\n
Bayes	.008%	0%	'\r\n': 0.0000, ': ': 0.0000, '\r\nHost: ': 0.0022, 'GET ': 0.0035, ' HTTP/1.1\r\n': 0.1108, '\xFF\xBF': 3.1517. Threshold: 1.9934

Fig. 3

Results of multiple polymorphic worms + noise and single worm + noise were similar for token subsequence signature while there were some false positives for the bayes signature. The argument was that polygraph can generate 100% efficient signature if there are more samples (50+) than if there are less samples.

Time

According to the paper, time is not a big constraint either for hierarchical signatures, because they were successful in deriving signature within an acceptable amount of time, 25 seconds. If the sample pool is noisy with comparatively less samples than this time may not be sufficient to derive an efficient signature.

2.1.4 Final Word – Conclusion

However above results showed an efficient approach, but it has a narrow scope. Resource utilization attacks and protocol based attacks are not analyzed. Moreover the assertion of the designers was , "we haven't analyzed general attacks but *polygraph-specific* attacks " , which is inline with their motivation of detecting polymorphic worms , but this technique should be incorporated with

protocol specific information to be more specific in detecting general attacks. A more rigorous testing with multiple set of worms could have better benchmarked the results.

2.2 Honeycomb

2.2.1 Concept

Honeycomb is a system that automatically generates signatures for malicious network traffic automatically by using pattern detection techniques and packet header conformance tests on traffic captured on honeypots so that we deal only with suspicious traffic. Honeycomb inspects traffic inside the honeypot at different levels in the protocol hierarchy, i.e. IP, TCP, UDP headers and payload data is examined. Honeycomb supports signatures for the Bro and Snort Network Intrusion Detection Systems and tries to ensure that the signature is narrow enough to capture precisely the characteristic aspects of the exploit it attempts to address and flexible enough to capture variations of the attack. Honeycomb is an extension of honeyd, a low interaction open-source honeypot. Honeypots are decoy computer resources that are set up for the purpose of monitoring and logging the activities of entities that probe, attack or compromise them. honeyd simulates hosts with individual networking personalities. It intercepts traffic sent to nonexistent hosts and uses the simulated systems to respond to this traffic. The honeycomb system tries to spot patterns in traffic previously seen on the honeypot: parts of flows in the traffic are overlayed and a Longest Common Substring (LCS) algorithm is used to spot similarities in packet payloads.

2.2.2 Technical Details

Some of the important aspects of the Honeycomb system are:

1) honeyd extension:

Honeycomb has added two new concepts to honeyd, a plugin infrastructure and event callback hooks. The former allows us to write extensions that remain logically separated from the honeyd codebase, and the latter provides a mechanism to integrate plugins into the activities inside the honeypot.

2) Signature creation algorithm

The diagram clearly illustrates the flow diagram for signature generation , we will not go in details , as it is not in scope of this paper. Readers are encouraged to look at the reference [1] for more detail information.

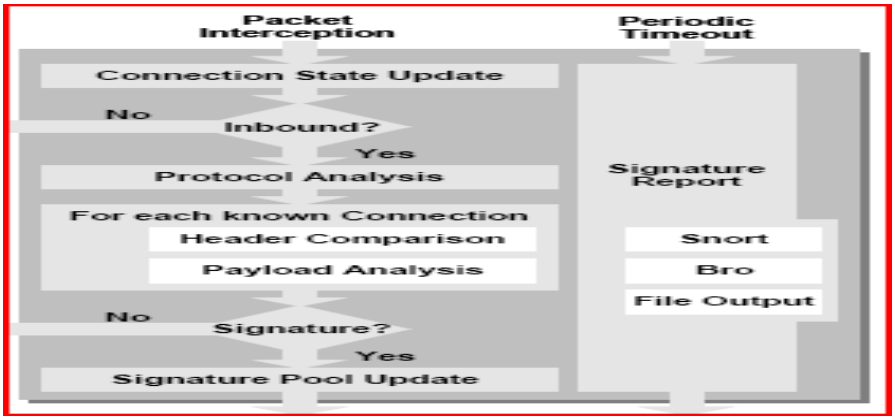

Fig . 4

3) Connection tracking:

Connections that have exchanged lots of information are potentially more valuable for detecting matches with new traffic. The system must prevent aggressive port scans from overflowing the connection hashtables which would cause the valuable connections to be dropped. Therefore, both UDP and TCP connections are stored in a two-stage fashion: Connections are at first stored in a "handshake" table and move to an "established" table when actual payload is exchanged.

4) Protocol analysis

Honeycomb performs protocol analysis at the network and transport layers for IP, TCP and UDP packet headers, using the header-walking technique previously used in traffic normalization. Instead of correcting detected anomalies, we record them in the signature, for example invalid IP fragmentation offsets or unusual TCP flag combinations. Honeycomb then performs header comparison with each currently stored connection of the same type (TCP or UDP). If the stored connection has already moved to the second level hashtable, Honeycomb tries to look up the corresponding message and uses the headers associated with that message. If any overlaps are detected (e.g., matching IP identifiers or address ranges), the analysis signature is cloned and becomes specific to the currently compared flows. The discovered facts are then recorded in the new signature.

5) Pattern Detection in flow content:

Honeycomb applies LCS algorithm to binary strings built out of exchanged messages using the following two methods:

Horizontal detection: Assume that the number of messages in the current connection after the connection state update is n. Honeycomb then attempts pattern detection on the nth messages of all currently stored connections with the same destination port at the honeypot by applying the LCS algorithm to the payload strings directly.

Vertical detection: Honeycomb also concatenates incoming messages of an individual connection up to a configurable maximum number of bytes and feeds the concatenated messages of two different connections to the LCS algorithm. Vertical detection also masks TCP dynamics: the concatenation suppresses the effects of slicing the communication flow into individual messages, which proved to be valuable.

6) Signature lifecycle:

If the signature report contains no facts at this point, processing of the current packet ends. The signature pool is implemented as a queue with configurable maximum size; once more signatures are detected that can be stored in the pool, old ones are dropped. The contents of the pool are reported at regular intervals. Honeycomb tries to reduce this by performing signature aggregation.

7) Signature output:

The comments of the pool are periodically reported to an output module that implements the logging of the signature records and are converted into Bro or pseudo-Snort format.

2.2.3 Parameters

Types of signature generated

Honeycomb uses pattern detection techniques and packet header conformance tests on honeypot captured traffic to generate signatures for use in both Bro and Signature IDSs.

Types of attacks

The system is not conducive in producing signatures for polymorphic worms. Honeycomb creates precise signatures for Slammer and CodeRed II worms.

Flow Pool

Honeycomb produces NIDS signatures automatically by analyzing traffic on a honeypot which provides the benefit of knowing that it is dealing with suspicious traffic.

Testing environment

Honeycomb was tested on an unfiltered cable modem in three consecutive sessions covering three days.

Results and Detection Rate

During the 24 hour period, 224 KB of traffic was captured which comprised of 557 TCP connections, 145 UDP connections and 27 ICMP pings. Honeycomb created 38 signatures for hosts that probed common ports. 25 signatures were created containing flow content strings.

Time

Honeycomb can run for extended time on quiet sites without problems. It deals with honeypot captured traffic so the system can perform at higher loads.

2.2.4 Final Word – Conclusion

However, more testing is required to evaluate system performance on busy sites with traffic that leads to increases amounts of connection state. The system is presently leading to large number of false positives when long identical byte sequences are in use. The system could benefit with the incorporation of a white list for benign sequences.

2.3 EarlyBird

2.3.1 Concept

EarlyBird as other signature generation algorithm assumes invariant bytes in polymorphic worms which are then used for signature generation. However with invariant bytes , EarlyBird also takes in account address dispersion , which means number of source and destination address in worm packet will contain uniform addresses because the worm will likely try to infect a number of machines on a network following in single domain. This address dispersion will be different from that produced by normal network traffic. Every unique substring is indexed in a prevalence table with an associated count value, also source and destination addresses are also indexed. Sorting the table on above parameters will segregate worm traffic from normal network traffic. EaryBird uses content sifting to filter normal traffic from malicious . It uses a hash to

maintain payloads of packets which appears at least X times in overall N packets sent during the interval. However for worms it may be possible that invariant bytes may take a small size less than a packet payload. Storing and indexing such smaller strings is addressed by this approach using polynomial incremental rabin fingerprints for small substrings. The essence is that for equal substrings rabin fingerprint will produce same fingerprint, which will help in updating the value of count for that string , appearing *Y* times. When this value of *Y* reaches value of X , this fingerprint is reported. EarlyBird generates a contiguous protocol dependent signature , with a single contiguous unique substring identifying a worm.

2.3.2 Technical Details

This approach uses direct bitmap for identifying address dispersion as follows :

Each content source is hashed to a bitmap and the corresponding bit is set. A threshold value is chosen , if no. of count bits exceeds the threshold than an alarm is raised indicating use of same source address many times. However this approach is memory efficient , but may have a bottleneck if no. of source addresses exceeds exponentially. To optimize counter estimation for each address scaled bitmap is implemented by sampling the range of hash space. The approach of EarlyBird is to use minimal memory and CPU requirement in generating signatures. Memory requirements are met by using Rabin fingerprints and scaled bitmap for address dispersion, but if a substring length is of length 20 bytes and payload exceeds normal 256 bytes, calculation of rabin fingerprints will effect CPU throttling. To optimize CPU , two approaches were identified :-

a. Using random samples from payload
b. Using dynamic value samples with some pre-specified pattern and sampling frequency

Use of random samples may miss invariant worm substring , and will delay the signature generation procedure. However using dynamic value sampling the probability of worm detection depends on sampling frequency , pattern type and length of the substring to be found.

Practical implementation of EarlyBird uses a 40-byte rabin fingerprint and a 32-bit CRC for packet hash. Each packet arrived is hashed ,either full payload or substrings ; port and protocol information are appended to the content. Hashes are indexed in address dispersion table only if an entry is found , or otherwise these contents are indexed in prevalence table. If threshold for content in prevalence table exceeds the threshold than a new entry is placed in the address dispersion table. If an entry already exists for the content , than dispersion table entries for source and destination addressed are updated, indicating the prevalence for the substring occurring in multiple flows of the same worm (polymorphic) . Thus on exceeding threshold , the content is

reported. Address dispersion table thus has all the entries which are candidates of malicious traffic.

EarlyBird System Design

The above algorithm is implemented using two major components : *Sensor* and *Aggregator.* Sensors are responsible for monitoring network traffic, content sifting and signature generation . Aggregator coordinates with the sensor by configuring signatures , and implementing client based blocking techniques. Aggregator is also responsible for reporting and administrative controls using PHP script for maintaining GUI for administrative access. This system can also generate SNORT compliant inline signatures for preventing intrusions. SNORT inline is an intrusion prevention tool which uses Iptables on unix operating system to block and drop malicious packets.

2.3.3 Parameters

Testing Environment

EarlyBird is designed and deployed at University of California , San Diego campus. It uses Cisco catalyst router to mirror all in-bound and out-bound traffic to sensors. Prevalence threshold is kept at 3 and address dispersion threshold to 30 source and 30 destinations. The rationale behind the threshold is if any content is prevalent and exceeds prevalence threshold, it is still not justified to qualify that as a worm candidate. This is because a substring can occur a multiple number of times between two distinct hosts. So it is necessary to track hosts as well as contents for qualifying a traffic to be malicious.

False positives

False positive rate can be efficiently checked by using a mix of data including noise , but in case of testing done for EarlyBird they have given no indication as what was their flow pool. Moreover as this approach uses source and destination addresses and port numbers to qualify a traffic, it is rather less probable to have more false positives , but more false negatives.

Detection Rate and Types of attack

Criteria for detection rate and false negatives were not clearly mentioned in the study. However testing environments does not indicate a qualitative measure for detection rate. As the signatures are tied to addresses and port numbers a simple evasion of the approach would be to spoof address and change port numbers frequently. As polymorphic worms are not static , it is probable that those signatures generated at initial detection will not be valid for multiple occurrence of the worm.

Sample results

Label	Service	Sources	Dests
SLAMMER	UDP/1434	3328	23607
SCAN-TCP-PORT22	TCP/22	70	53
MAIL-HEADER-FROM	TCP/25	12	11
SMB-139	TCP/139	603	378
SMB-445	TCP/445	2039	223
HEADER-TCP-CLOSE	TCP/80	33	136
MAIL-HEADER-FROM2	TCP/25	13	14
PROTOCOL-HEADER-EXT	TCP/80	15	24
BLASTER	TCP/135	1690	17
OPASERV-WORM	UDP/137	180	21033
SMB-445-SIG2	TCP/445	11	145

Fig. 5

The above results shows hat worms specific , which are RAT's (Remote administrative tools) have a binding with port numbers, which may not be the case with a more general attack. The worms for which results are shown in Fig. belongs to the category of RAT's . A more rigorous testing with normal attack patterns would have better benchmarked the results.

Types of signatures

Earlybird is capable of generating SNORT compliant signatures. SNORT signatures exists in token subsequence format, which protocol information appended at the start. Thus EarlyBird is able to generate signatures which include protocol information , destination and source address and ports.

Sample signature

" drop tcp $HOME_NET any -> $EXTERNAL_NET 5000
(msg:"2712067784 Fri May 14 03:51:00 2004";
rev:1; content:"|90 90 90 90 4d 3f e3 77 90
90 90 90 ff 63 64 90 90 90 90 90|";) "

2.3.4 Final Word- Conclusion

EarlyBird uses a complex computational concept in qualifying signatures which proved effective in many cases of identified worms. However the assumption of invariant bytes to generate signatures is still an overrated assumption made by many systems discussed including EarlyBird. It is possible as in case of rule based IDSes to evade EarlyBird using packet manipulation, because assumptions made by this approach binds the working with some specific traffic pattern , which needs to be sufficiently different running on different ports using uniform address dispersion. An improvement in this approach could be to use normal behavior against what is monitored which

appears on wire. Such knowledge will help in making Earlybird scalable to more general attacks.

3. Signature Generation With Attack Detection

3.1 Autograph

3.1.1 Concept

Autograph is a system that automatically generates signatures for novel internet worms that propagate using TCP support. It does so by analyzing the prevalence of portions of flow payloads. Autograph is extended to share port scan reports among distributed monitor instances to speed the generation of novel worm signatures. A signature is a 2-tuple (IP-protocol number is not considered as we deal with worms that propagate over TCP) consisting of (dst-port, byte seq) tuples where content based filtering is used to obtain a match when byte seq is found within the payload destined for dst-port. Autograph builds a system that automatically, without fore-knowledge of a worm's payload or time of introduction, detects the signature of any worm that propagates by randomly scanning IP addresses. It is assumed that the system monitors all inbound network traffic at an edge network's DMZ. The evaluation of Autograph is concerned with two important themes, firstly the tradeoff between early generation of signatures for new worms and the specificity (low false positives) of the generated signatures and secondly the utility of distributed collaboratively monitoring in speeding detection of a novel worm's signature after its release.

3.1.2 Technical Details

Some of the key design goals are illustrated in brief:

a. Signature quantity and length- A flow must be compared to all signatures known for its IP protocol and port. Fewer signatures speed matching. The cost of signature matching is proportional to the length of the signature, so short signatures may be preferred.

b. Robustness against polymorphic worms- A strongly polymorphic worm is one whose successive payloads share only very small byte subsequences in common. If a polymorphic worm does not change one or more relatively long subsequences across its variants, an efficient signature detection system will generate signatures that match these invariant subsequences, and thus minimize the number of signatures required to match all the worm's variants.

c. Timeliness of detection- Patching of infected hosts is more effective the earlier it is begun after the initial release of a new worm, and that in practical deployment scenarios, patching must begin quickly (before 5% of vulnerable hosts become infected) in order to have hope of stemming an

epidemic such that no more than 50% of vulnerable hosts ever become infected.

The three modules in the Autograph architecture are shown in fig. 6 :

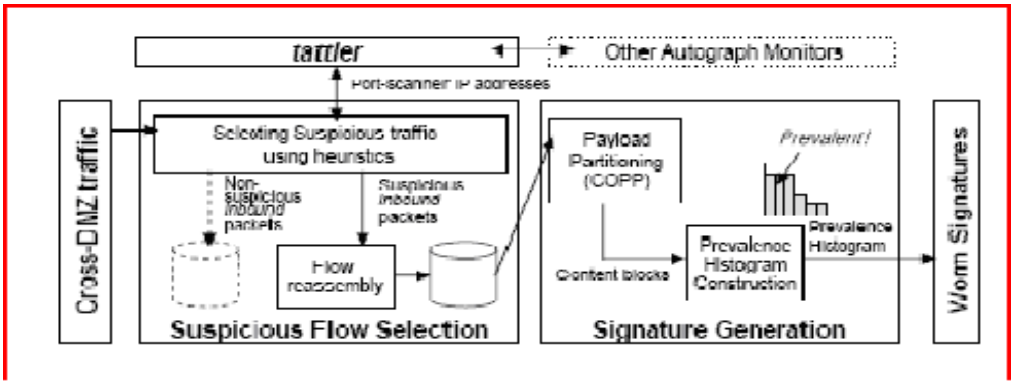

Fig. 6.

Flow Classifier:

There are two main stages in a single Autograph monitor's analysis of traffic. First, a suspicious flow selection stage uses heuristics to classify inbound TCP flows as either suspicious or non-suspicious. After classification, packets for these inbound flows are stored on disk in a suspicious flow pool and non-suspicious flow pool, respectively. Further processing occurs only on payloads in the suspicious flow pool. Thus, flow classification reduces the volume of traffic that must be processed subsequently. Autograph performs TCP flow reassembly for inbound payloads in the suspicious flow pool. Autograph stores the source and destination addresses of each inbound unsuccessful TCP connection it observes. Once an external host has made unsuccessful connection attempts to more than s internal IP addresses, the flow classifier considers it to be a scanner and writes the IP address to the suspicious pool, until the IP addresses are removed after timeout t. Autograph reassembles all TCP flows in the suspicious flow pool and every r minutes it considers initiating signature generation. This happens when the pool contains more than a threshold no. of flows. The resulting reassembled payloads are analyzed in Autograph's second stage signature generation.

Content based Signature Generation:

Autograph segregates flows by destination port for signature generation. It involves analysis of the content of payloads of suspicious flows to select sensitive and specific signatures. The two main traits of worm traffic content commonality and magnitude of traffic volume suggest that analyzing the frequency of payload content should be useful in identifying worm payloads. During signature generation, Autograph measures the frequency with which non-overlapping payload substrings occur across all suspicious flow payloads, and proposes the most frequently occurring substrings as candidate signatures. It

divides each suspicious flow into smaller content blocks and counts the no. of suspicious flows in which each content block occurs and this is termed as its prevalence. When all worm flows contain a common, worm specific byte sequence, it will be observed in many suspicious flows and will have high prevalence.

Tattler:

The tattler shares suspicious source addresses among all monitors, toward the goal of accelerating the accumulation of worm payloads. Autograph seeks to allow monitors to announce to the others the (IP-addr, dst-port) pairs they have observed port scanning themselves, to limit the total bandwidth of announcements sent to a multicast group and to allocate announcement bandwidth relatively fairly among monitors. In the tattler protocol, each announcement a monitor makes contains between one and 100 port scanner reports of the form (src-IP, dst-port). Monitors only announce scanners they've heard themselves. Hearing a report from another monitor for a scanner suppresses announcement of that scanner for a refresh interval. After a timeout interval, a monitor expires a scanner entry if that scanner has not directly scanned it and no other monitor has announced that scanner. running a distributed population of Autograph monitors holds promise for speeding worm signature detection in two ways: it allows the "luckiest" monitor that first accumulates sufficient worm payloads determine the delay until signature detection, and it allows monitors to chatter about port-scanning source addresses, and thus all monitors classify worm flows as suspicious earlier.

3.1.3 Parameters

Types of signature generated

Autograph can generate worm content signatures for use in content based filtering. As discussed above content bases signatures slow down the spreading of a worm effectively. It publishes signature byte patterns in Bro's signature format, for direct use in Bro

Types of attacks

Autograph generates signatures by analyzing the prevalence of portions of flow payloads in real time and hence it generates signatures early. As the design is catered to generate signatures for polymorphic worms that do not change one or more relatively long subsequences across its variants, Autograph can generate small content block sizes which are resilient to polymorphic attacks. Autograph is susceptible to DoS attacks if it tries to reassemble every incoming suspicious flow. If a worm propagates using a hit list instead by scanning IP addresses, Autograph fails to work.

Flow Pool

Autograph contains both a suspicious and a non suspicious flow pool. The processing of the payloads in subsequent stages occurs only on the suspicious flow pool. Thus, flow classification reduces the volume of traffic that must be processed.

2.2.4 Final Word – Conclusion

The offline evaluation of Autograph on real DMZ traces reveals that it can generate sensitive and specific signature sets that exhibit high true positives and low false negatives. However, the results of running Autograph in an online setting where the system generates signatures periodically using the most recent suspicious flow pool would help in further analyzing Autograph's overall effectiveness. The susceptibility of Autograph to DoS attacks and to worms that propagate using hit list must also be looked into. Moreover, further exploration means beyond sharing port scan reports amongst distributed monitor instances using source IP addresses must considered to improve speed and quality of signature detection and generation. To conclude, Autograph does have minor limitations but is effective to automatically generate signatures against novel worms.

3.2 P.A.D.S

3.2.1 Concept

Passive Asset Detection System (P.A.D.S) is a network asset detection system being developed under the open source license by Matt Shelton. The system is being designed to supplement active system scanners such as Nmap and Nessus with a more passive and less intrusive system . P.A.D.S is designed to be used along with current Intrusion Detection Systems (I.D.S) to identify the different programs or services running on a network and act as an additional source of information for any I.D.S alerts. The system uses signature based detection rules to detect the assets within a network. Since the system is designed to be passive it only uses the detection rules on incoming data to identify the source and map out the network and its assets. The system does not perform any active scanning at any point during its program cycle.

Active scanners such as Nmap rely on active access to the network they are monitoring. This places many limitations on their usability since some Intrusion Detection Systems are only used to monitor a given network as such may not have actual access to it. Passive scanners such as P.A.D.S only rely on reviewing incoming data from a source to identify the type of asset, as such making them more plausible in such scenarios.

According to the program developer, Matt Shelton, the program is designed to achieve the following goals.

Passive. The program records the traffic on the given network and identifies the various assets based on its signature detection rules. Since the program does not perform any active scanning, there is no data sent from the system to the network.

Portable. The system is designed to be easily placeable on remote systems. The system does not require any external library sources except for those associated with libcap. Libcap is used to narrow down the scope of the monitoring using filters such as port specification.

Lightweight. The system logs all its data to a simple CSV file. This removes the need for intensive storage means such as databases or other external data repositories to be installed on the local machine. Currently the author plans to develop an optional feature for the system to provide data aggregation and correlation for network administrators.

P.A.D.S in a Network

The P.A.D.S program can be placed on any node in the network which receives data from the network which needs to be monitored for asset detection. The system uses sensors already present on the node to compare the data against its list of signatures. If a match is found for an asset which is not currently present in the CSV log file the program appends the new asset to it. The program also maintains a simple log of the number of packets coming in to the node.

3.2.2 Technical Details

The Passive Asset Detection System is being developed for All BSD Platforms (Including FreeBSD/NetBSD/OpenBSD/Apple Mac OS X), All POSIX (Linux/BSD/UNIX-like OSes) and Linux platforms.

Currently the program looks at three types of incoming traffic.

Transmission Control Protocol (TCP) This set is used to detect commonly used protocols such as data transfer through Port 22 or the OpenSSH 3.8.1

Address Resolution Protocol (ARP) Used to detect the source from which data is being received. For example, 00:B0:D0:92:2F:17 (Dell Computer Corp.)

Internet Control Message Protocol (ICMP) Used to detect error messages being sent over a network. Additional detection rules are being developed to add more functionality and details to the scanning abilities of the program. Most recently the author is looking to add OS identification rules to the system. The program uses a current set of signatures and compares incoming data with it.The program appends any signature matches to a dynamically updated CSV log file which can be reviewed by the network administrator. Depending on the type of protocol the incoming data uses the program can

provide details such as IP, DNS, MAC addresses of the sender or the port of incoming traffic and type.
The system is dependent on I.C.M.P echo packets. If any given host service or asset does not reply to I.C.M.P echoes the system will be unable to detect them. The P.A.D.S system can detect services which are present on specific networks to which the system is connected.

3.2.3 Parameters

Generated output

The program by default generates a comma separated log file. Each line in the log corresponds to a unique asset/service found on the monitored network. The data corresponding to the service depends on the type of protocol is uses. As mentioned in the technical details section, the program only detects services and assets which reply to I.C.M.P packets.

Types of assets detected

The program can only detect assets which are present in its signature list. The signature list contains templates for protocols for incoming data. The command line configuration can be used to filter out which signatures the program uses to detect incoming traffic. The program can also be configured to monitor only the data coming from specific networks.

Flow pool

The program compares every incoming data packet from the node sensor with its signatures list. The only exceptions applied are those specified at the command line configuration.

Testing environment

Since P.A.D.S is still in its beta stage there are very limited tests done on the program usage. In the few tests done by the open source community at Sourceforge, the program was run on Red Hat 9.0 and FreeBSD 5.2.1. The tests were done using both the default mode where the program examines every data packet to its complete list of signatures and using filters for specific networks and ports.

Results and detection rate

The program recognized all the data packets in its signature list. There is currently no logging for data packets which do not match any template in the signature list. The program uses I.C.M.P echo replies for its detection as such in networks or sub networks where there were no services or I.C.M.P echo

replies were disabled, the program did not provide any type of output to let the user differentiate between the two.

Result

A sample test result

```
10.0.0.1,0,0,ARP,00:06:25:78:20:75,1092511120
10.0.0.81,0,0,ARP,00:50:da:5a:2d:ae,1092511122
10.0.0.225,0,0,ARP,00:06:25:12:57:0e,1092511125
10.0.0.1,0,1,ICMP,ICMP,1092511146
10.0.0.81,0,1,ICMP,ICMP,1092511149
10.0.0.83,0,1,ICMP,ICMP,1092511149
10.0.0.83,22,6,ssh,OpenSSH 3.8.1 (Protocol 2.0),1092511226
10.0.0.85,0,0,ARP,00:0c:29:ba:1e:02,1092511274
10.0.0.85,0,1,ICMP,ICMP,1092511274
10.0.0.83,0,0,ARP,08:00:20:a0:14:a5,1092511279
10.0.0.85,22,6,ssh,OpenSSH 3.5p1 (Protocol 1.99),1092511282
10.0.0.1,80,6,unknown,unknown,1092511345
10.0.0.225,80,6,www,Ubicom/1.1,1092511364
10.0.0.83,80,6,www,Apache 1.3.31 (Unix),1092511493
```

3.2.4 Final Word – Conclusion

The above result is a sample of the raw output the program appends to its CSV log file. The program in its current stage is very situational in its use. The data provided by the data is insufficient to completely supplement active scanners currently available. Once the program receives additional signature detection rules and OS identification module, rigorous testing in a controlled environment can provide a better view of viability of the program.

3.3 PAYL

3.3.1 Concept

PAYL is an intrusion detection method based on the detection of anomalies in the payload. One of the first principles behind the use of the PAYL detection system is the launch of zero-day attack which will have packet content the victim has never seen before. Zero-day attacks/worms spread very quickly and infect a large number of systems. If there is no signature detection available to stop this attack at one place, then it also means that it is likely that detection of the attack is unlikely at other sites as well. This means wide spread infestation as signatures to detect and stop these attacks take longer to create than the time it take for the worm to spread. The PAYL sensor is created to be able to detect zero day attacks right at the time of launch or soon afterward. While most attacks can be detected using packet header there are some attacks that deliver the normal content while also transporting malicious code. These attacks deliver packets that have unusual

payload when compared to the prior content of packets flowing into the network. While other detection methods use packet headers or other means, PAYL uses payload detection, so even if the normal content is delivered, if an anomaly is exists in the content, it can also be detected.

While there are other detection systems based on packet payload based on signatures, they cannot be used for new and unknown attacks, for which signatures have not been created. PAYL through the detection on anomalies is able to detect even new attack through real time. The new method used for detection is not the scanning or probing of information but an ingress/egress correlation. The main idea is that the victim will send out similar content that it was infected with to others so that infestation can spread. This helps is the detection of worms from the moment it tries to propagate itself and will stop wide spread infestation. Not all of the packets are to be checked for malicious code, but suspects inbound packets that contain data different from the normal packets. One of the benefits of using this method is that while checking for worms, if one is detected, a signature is created by the ingress/egress methods that can be sent to other networks to stop infestation.

3.3.2 Technical Details

PAYL Content Profile

PAYL uses the principle that the packets from new attacks will have unusual content from the normal. So using the packet content from the sensor, a normal profile is created that holds the sites unique packet flow. The profile is created using data that was seen during the training phase of the sensor. The content data used to create the profile must be very accurate and efficient. PAYL uses a language independent method that requires no parsing, interpretation and emulation. A distribution of n grams is taken from the networks packet datagram for analysis. The n grams are a sequence of n adjacent byte from the payload. A sliding window is use to pass over the entire payload and record the frequency for each n-gram. Then a distinct model based on port and packet length is taken to produce a total and effective model of the site actual flow.

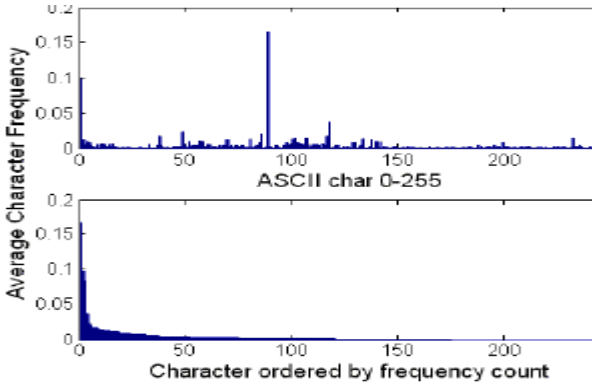

Figure – 6

GET./default.ida?XXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXXXXXXXXXXXXXXXXXXXX
XXXXXX%u9090%u6858%ucbd3%u7
801%u9090%u6858%ucbd3%u7801%u
9090%u6858%ucbd3%u7801%u9090%
u9090%u8190%u00c3%u0003%u8b00
%u531b%u53ff%u0078%u0000%u0

Figure – 7

Figure 6 shows the Code Red II packet while Figure 7 shows the distribution of the byte value. Using the data from this a z-string is created, which serves as a representative of the entire distribution. The main function of the z-string is to provide privacy when data has to exchanged between two domains without showing the real content of the packet. PAYL uses the Mahalanobis distance, which weights each variable, the mean frequency of the n-gram, and its statistics. If the distance of the test is greater than the threshold, then an alert is send showing detection of malicious code. The threshold is distinct for each centroid but the initial setting threshold is chosen, then using updates, the new threshold is calculated. Once this is done, PAYL is ready to enter the detection mode.

Ingress and Egress Traffic Correlation

When PAYL detects strange packets on an incoming port, it places an alert and put it into a suspect list. Then if any of the outgoing packets contain that data going to the same port a comparison is performed on the contents of the packet for similarity and a score obtained. Then the score is compared to a threshold and if it is higher, the traffic for that content. This is an improvement from other similar forms in that PAYL only stops traffic of the suspect content and not all traffic all together. The form of string comparison is to be noted as this is the check for worms. There are three different type that PAYL uses commonly: String equality, longest common substring, and longest common subsequence. String equality is the most intuitive approach. It requires and exact match in the suspect packet so it is good at reducing false positives. Longest common substring is a less exact approach but does not require any payload manipulation. The main shortcoming is the computation overhead compared to string equality. Longest common subsequence has the ability to detect polymorphic worms but it also can detect more false positives. The type of methods used should be dependent on the network as we have to looks at computational cost and time.

3.3.3 Parameters

Types of signature generated

An important benefit of the ingress/egress correlation is the creation of worm signatures automatically. The similarity score produces a substring that represents the common part of the malicious code in both the incoming and outgoing traffic; this can serve as the worn signature filter. Another methods is the fact that since they have already been identified as malicious code, if similar payload are being send to the same host from other host, the common parts could be used to represent the worm signature. An example is shown in figure 3, which shows PAYL generated signature for the CodeRed II worm. If the LCSeq method of checking is used, the even polymorphic worm could be detected as the core exploit is usually the same. So by using the ingress/egress methods we can not only detect worms but at the same time produce a worm signature which can be given to other sites in the network.

Flow Pool

PAYL monitors all the packets but does no probing or scanning unless it is detected as a unusual suspect. The suspect packet is stored in a buffer to be checked against outgoing traffic . If a similar packet is send to the same port then the payload of the suspected packets are scanned for worms by comparing the ingress and egress packets. Based on the type of string comparison used the rate of false positives changes, but with string equality, the rate of false positives is minimized.

Testing environment

In order to test the effectiveness of PAYL, a test bed was set up that used three datasets captured from real traffic using known worms available for research. Each test dataset a clean set of packets was used that were free from all know worms. Then the same set of worm traffic was put into each of the test datasets. Thus a true control environment was created for each of the three sets. The worms that were put in were the Code Red , Code Red II, WebDAV, and another worm that exploits the Windows media service. The worm set was then inserted at random places in the data set.

Results and Detection Rate

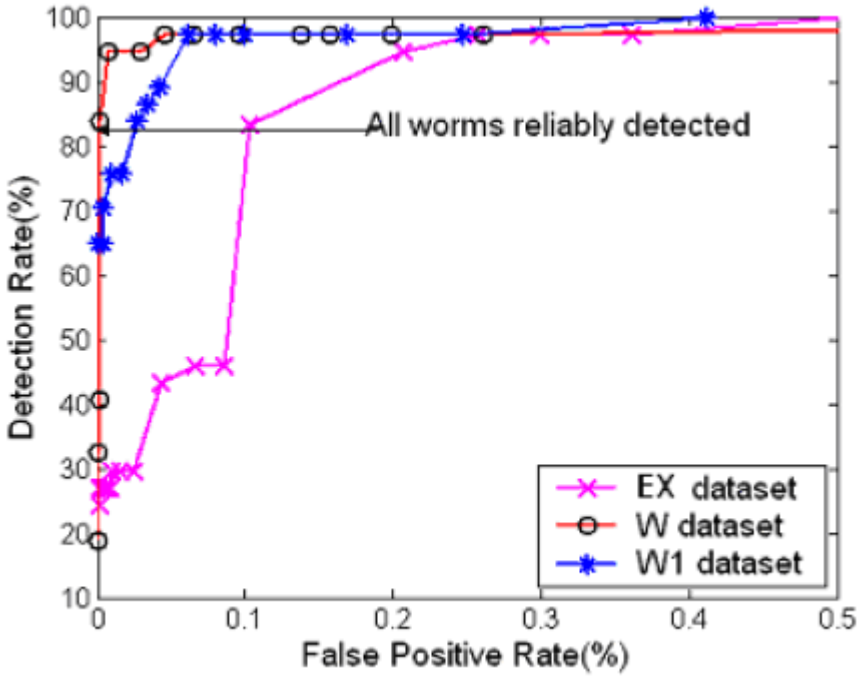

Figure-8 Results & Detection Rate

The numbers of packets is used to base the detection rate and the rate of false positives. The test contained 40 worm packets. As figure 4 shows, PAYL was able to successfully detect the entire worm, with a very low false positive rate. The lines

on the figure show when all four worms were detected. So as the false positive rate was increased slightly, a huge increase in the detection rate can be seen. All of the different sites contained different payload distributions account for the varying detection rates, but even with this, PAYL was able to detect the worms at a very fast rate.

Time

The time to detect the instance of new attacks is very small. In some cases the first instance of the attack itself could be detected. Even with new worms by using the ingress/egress method the possibility of detection is very high as it check the payload content. So the time of detection is very small, and when comparing to other forms of detection proves to be very efficient. Using the minimization in time, signatures could be sent to other sites to help prevent them from infestation.

3.3.4 Final Word – Conclusion

The preliminary results for PAYL look very good and show an efficient method for protection of worms and zero-day attacks. As the results show, PAYL was able to efficiently detect the 4 worms with a very small false positive rate. Still more research needs to be done as the test was small, and repetitions must also be tested to see if similar results can be achieved. The test also did not use unknown worms so the feature of the ingress/egress to create new worm signatures for unknown viruses has still to be test. But the results show so far seem to hold great promise for the future as an efficient way of protecting cyber space.

4. Final conclusion

Approach specific conclusions and recommendations are discussed in every methodology, however there is something in common in all those approaches which I like to focus on

1. Although, systems analyzed in our report proves their effectiveness, but it seems that they aren't deployed and tested in commercial environment.

2. A best test for all these systems would be to put them in balanced testing environment on same testing beds to review and analyze their true results.

3. Extensive testing is needed before real world deployment.

References

[1] Attack Detection and Signature Generation –The NoAH Project
http://fp6-noah.org/publications/deliverables/D1.2.pdf

[2] James Newsome, Brad Karp and Dawn Song. Polygraph: Automatically Generating Signatures for Polymorphic Worms.
www.cs.berkeley.edu/~dawnsong/papers/polygraph.pdf

[3] Sébastien Chainay and Karima Boudaoud. A Model for Automatic generation of Behaviour-based worm signatures.
www.laas.fr/METROSEC/MonAM2006.pdf

[4] Ke Wang, Gabriela Cretu and Salvatore J. Stolfo. Anomalous Payload-based Worm Detection and Signature Generation.
www1.cs.columbia.edu/ids/publications/raid-camerav.pdf

[5] Dimitry Averin. Autograph: Toward Automated, Distributed Worm Signature Detection
www.cs.cmu.edu/~bkarp/autograph-usenixsec2004.pdf

[6] SNORT – snort.org .An Open Source Network Intrusion Detection System.

[7] S. Singh, C. Estan, G. Varghese, and S. Savage. Automated worm fingerprinting.
http://www.cs.ucsd.edu/~savage/papers/OSDI04.pdf