Detecting Zero-Day Attack Signatures using Honeycomb in a Virtualized Network

Reshma R. Patel
Information Technology Department,
L.D.College of Engineering,
Ahmedabad, India.

Chirag S. Thaker
Information Technology Department,
L.D.College of Engineering,
Ahmedabad, India.

Hemant B. Patel
Software Engineer-Team Leader,
Alferez Pvt. Ltd Vadodara,
India.

ABSTRACT

Self-propagating malware, such as worms, have prompted cyber attacks that compromise regular computer systems via exploiting memory-related vulnerabilities which present threats to computer networks. A new generation worm could infect millions of hosts in just a few minutes, making on time human intrusion impossible. The new worms are spread over the network on regular basis and the computer systems and network vulnerabilities are growing enormously. Here we also facing the problem of automatically and reliably detecting previously unknown attacks which are known as zero-day attack.In this paper, I have described the use of the Honeycomb to detect Zero-day attack in Virtualized network. A method to automatically generate signatures using the proposed detection system is presented. The attack signatures are detected and scanned through the system. Honeycomb is a host-based intrusion detection system that automatically creates signatures. It uses a honeypot to capture malicious traffic targeting dark space, and then applies the Longest Common Substring (LCS) algorithm on the packet content of a number of connections going to the same services. The computed substring is used as candidate worm signature. Honeycomb is well suited for extracting string signatures for automated updates to a firewall.

General Terms

Zero-Day Attack Signatures detection.

Keywords

Zero-Day attack, Honeycomb, Malware, Automatic Signature Generation

1. INTRODUCTION

Self-propagating malware, such as worms, have prompted a wealth of research in automated response systems. We have already encountered worms that spread across the Internet in as little as ten minutes, and researchers claim that even faster worms can be realised. For such outbreaks human involvement is too slow and automated response systems are needed. Important criteria for such systems in practice are: (a) reliable detection of a wide variety of zero-day attacks, (b) reliable generation of signatures that can be used to stop the attacks, and (c) cost-effective deployment [1]. *Wikipedia* defines 'zero-day virus' as 'a previously unknown computer virus or other malware for which specific anti-virus software signatures are not yet available'. Security should protect a computer from the effects of any malicious attack while staying completely invisible. Honeypots and Intrusion detection systems offer different tradeoffs between accuracy and scope of attacks that can be detected. A honeypot is a device or service that operates in a network and waits for any form of wicked or malicious interaction to be initiated with it.All interaction with a honeypot is closely monitored, as analysis of the interaction can provide information concerning vulnerabilities, worm propagation, targeted ports and a detailed attack model in the event of a full compromise. Intrusion detection is a set of techniques and methods that are used to detect suspicious activity both at the network and host level. Intruders have signatures, like computer viruses, that can be detected using software. Based upon a set of signatures and rules, the detection system is able to find and log suspicious activity and generate alerts.

2. PROBLEM

CPU, memory, and storage have improved significantly by the time. Software has not attached the full latent of available hardware, but it has grown in size and complexity. Hence, complex software frequently contains programming errors that reveal themselves as crashes or unexpected behavior. Fault distribution studies show that there is a correlation between the number of lines of code and the number of faults. To quantify this, it is approximated that code contains 6-16 bugs per 1000 lines of executable code. Attackers are often able to utilize certain types of program faults to evade security measures to protect a system. Reports by organizations such as SANS, and various CERT show that there is large number of such vulnerabilities.
Zero-day worms are a serious wide-scale threat among large numbers of replicated vulnerable systems. If any standard signature-based detector is unsighted to a zero-day attack, than all installations of that same detector are also blind to the same attack. Here one has to consider the problem of accurately detecting these "zero-day" attacks upon their very first appearance. Some attacks exploit the vulnerabilities of a protocol; others seek to survey a site by scanning and probing. These attacks can often be detected by analyzing the network packet headers, or monitoring the connection attempts and traffic volume.

2.1 Malware

Software Errors

Software errors, commonly referred to as bugs, have been the primary cause of most security vulnerabilities. They mainly arise from mistakes made by developers when coding programs, or can be the result of faulty designs. Less frequently, bugs can be introduced by a compiler that produces incorrect binary code even when given sound source code.

Buffer Overflows
A frequently encountered memory access error that results in storing data in a different location than the one intended by the programmer is a buffer overflow. Such errors usually occur when copying data between buffers without checking their size.

Format String Errors
A less common memory error can occur when an invalid format string is used with the printf(format, ...) family of functions. These functions produce string output that can be printed in standard output, or written to a string buffer or a file. The output is created according to the string in the format argument. The function accepts a variable number of arguments, which are stored in the stack. As the format string is processed, the arguments are retrieved from the stack to produce the output.

Worm
A worm is a program that propagates across a network by exploiting security awes of machines in the network. The key difference between a worm and a virus is that a worm is autonomous. That is, the spread of active worms does not need any human interaction. As a result, active worms can spread in as fast as a few minutes. The propagation of active worms enables one to control millions of hosts by launching distributed denial of service (DDOS) attacks, accessing confidential information, and destroying/corrupting valuable data. Accurate and prompt detection of active worms is critical for mitigating the impact of worm activities.
A worm is a program that propagates across a network by exploiting security awes of machines in the network. The key difference between a worm and a virus is that a worm is autonomous. That is, the spread of active worms does not need any human interaction. As a result, active worms can spread in as fast as a few minutes. The propagation of active worms enables one to control millions of hosts by launching distributed denial of service (DDOS) attacks, accessing confidential information, and destroying/corrupting valuable data. Accurate and prompt detection of active worms is critical for mitigating the impact of worm activities.

Self-propagating Malware
A particularly malicious threat against computer systems is that of self-propagating malware or worms. Internet worms such as CodeRed, Blaster, and Sasser have created havoc in the past, while recently the Conficker worm has also made the news on various occasions by infecting various high-profile targets. Worms are malicious code that use various infection techniques to compromise systems, and are able to self-replicate by locating and compromising new targets without the user taking any action. Table 1 shows GFI Software has announced the top 10 most prevalent malware threats for the month of February 2011as detected by scans performed by its anti-malware solution, VIPRE Antivirus, and its antispyware tool, CounterSpy.

Payload The program that implements the desired functionality of any malware, besides the infection of the target, is the payload. The payload of an attack is also called shellcode for historical reasons, as it was frequently used by attackers to acquire a remote shell on the compromised system.

2.2 Zero-Day Attack

Wikipedia defines 'zero-day virus' as 'a previously unknown computer virus or other malware for which specific anti-virus software signatures are not yet available'. According to *Wikipedia*, 'a zero-day attack or threat is a computer threat that tries to exploit computer application vulnerabilities that are unknown to others or undisclosed to the software developer'. There is also a notion of a *vulnerability window* which is the time between the first exploitation of vulnerability and when software developers start to develop a countermeasure to that threat. These definitions evaluate time points such as the attack release and the moment when the very first easing is available.
In the field of Anti-Virus products the test of zero-day protection is usually performed by using so-called proactive testing methodology (also known as *retrospective testing*). This involves 'freezing' a product (creating a snapshot and subsequently denying the product the ability to receive updates) and then testing detection over attacks which appeared after the freeze point. In this scenario the frozen product will only face unknown threats and therefore all the reactive capabilities will be excluded from the test.

3. DEFENCES

As documented by SANS, "Vulnerabilities are the gateways by which threats are manifested" .In other words, a system compromise can occur through a weakness found in a system. A Vulnerability assessment is a search for these weaknesses/exposures in order to apply a patch or fix to prevent a compromise. There are two points to consider:
Many systems are shipped with: known and unknown security holes and bugs, and insecure default settings (passwords, etc.).
Much vulnerability occurs as a result of misconfigurations by system administrators.
Ways to counteract these conditions include:
1) Creating and surviving by baseline security standards,
2) Installing vendor patches (when appropriate),
3) Vulnerability scanning,
4) Subscribing to and abiding by security advisories,
5) Implementing perimeter defenses, such as firewalls and router ACLs,
6) Implementing intrusion detection systems and virus scanning software.
There are several methods that are used to find new security vulnerabilities:
- Source code analysis
- Binary file analysis
    - Static analysis
    - Dynamic (runtime) analysis
- Runtime analysis of API functions
- Fuzzing methods (fault injection) and
- Hybrid methods (various combinations of above methods).

3.1 Intrusion Detection System

Intrusion detection is a set of techniques and methods that are used to detect suspicious activity both at the network and host level. *Intrusion Detection System* or IDS is software, hardware or combination of both used to detect intruder activity. Snort is an open source IDS available to the general public. Intrusion detection systems fall into two basic categories:

- Signature-based intrusion detection systems
- Anomaly detection systems.

Signature-Based intrusion Detection System:
Intruders have signatures, like computer viruses, that can be detected using software. You try to find data packets that contain any known intrusion-related signatures or anomalies related to Internet protocols. Based upon a set of signatures and rules, the detection system is able to find and log suspicious activity and generate alerts.
Anomaly Detection System:
Anomaly-based intrusion detection usually depends on packet anomalies present in protocol header parts. In some cases these methods produce better results compared to signature-based IDS.

Signature:
Signature is the pattern that you look for inside a data packet. A signature is used to detect one or multiple types of attacks. For example, the presence of "scripts/iisadmin" in a packet going to your web server may indicate an intruder activity. Signatures may be present in different parts of a data packet depending upon the nature of the attack. For example, you can find signatures in the IP header, transport layer header (TCP or UDP header) and/or application layer header or payload.
Usually IDS depends upon signatures to find out about intruder activity. Some vendor-specific IDS need updates from the vendor to add new signatures when a new type of attack is discovered.

Terminology

- Alert/Alarm: A signal suggesting that a system has been or is being attacked.
- True Positive: A legitimate attack which triggers an IDS to produce an alarm.
- False Positive: An event signaling an IDS to produce an alarm when no attack has taken place.
- False Negative: A failure of an IDS to detect an actual attack.
- True Negative: When no attack has taken place and no alarm is raised.
- Noise: Data or interference that can trigger a false positive.
- Site policy: Guidelines within an organization that control the rules and configurations of an IDS.
- Site policy awareness: An IDS's ability to dynamically change its rules and configurations in response to changing environmental activity.
- Confidence value: A value an organization places on an IDS based on past performance and analysis to help determine its ability to effectively identify an attack.
- Alarm filtering: The process of categorizing attack alerts produced from an IDS in order to distinguish false positives from actual attacks.
- Attacker or Intruder: An entity who tries to find a way to gain unauthorized access to information, inflict harm or engage in other malicious activities.
- Masquerader: A user who does not have the authority to a system, but tries to access the information as an authorized user. They are generally outside users.
- Misfeasor: They are commonly internal users and can be of two types:
    - An authorized user with limited permissions.
    - A user with full permissions and who misuses their powers.
- Clandestine user: A user who acts as a supervisor and tries to use his privileges so as to avoid being captured.

3.2 Honeypot

Honey pots are systems used to lure hackers by exposing known vulnerabilities deliberately. Once a hacker finds a honey pot, it is more likely that the hacker will stick around for some time. During this time one can log hacker activities to find out his/her actions and techniques. This information can be used later on to harden security on actual servers.
High-interaction honeypots consist of a real OS and applications running on hardware or under a VM whereas *low-interaction honeypots* expose virtual OS and services to attackers. Multiple hosts can be simulated by a single low-interaction honeypot using forged network stacks to simulate different OS, and scripts that perform simple protocol handling for simulated services. Honeypots are deployed to handle all or part of the unused IP address space in the network.
The common services running on Honeypot are, like Telnet server (port 23), Hyper Text Transfer Protocol (HTTP) server (port 80), and File Transfer Protocol (FTP) server (port 21) and so on. The honey pot is placed close to production server to lure the attacker so that the attackers can assume it as for a real server. Firewall and/or router is configured to redirect traffic on ports to a honey pot where the intruder assumes connecting to a real server. The alert mechanism is created so that when honeypot is compromised, the alarm is triggered. The log files is kept on other machine so that when the honey pot is compromised, the hacker does not have the ability to delete these files.

Virtual Honeypot
A virtual honeypot is simulated by another machine. Virtual honeypots are more flexible and scalable, since only a single machine can simulate many virtual honeypots that host different operating systems and services.

Honeyd
Honeyd is a framework for virtual honeypots that simulates computer systems at the network level. Honeyd supports the IP protocol suites and responds to network requests for its virtual honeypots according to the services that are configured for each virtual honeypot. To simulate real networks, Honeyd creates virtual networks that consist of arbitrary routing topologies with configurable link characteristics such as latency and packet loss.

Subsystem Virtualization
Honeyd supports service virtualization by executing UNIX applications as subsystems running in the virtual IP address space of a configured honeypot. This allows any network application to dynamically bind ports, create TCP and UDP connections using a virtual IP address. Subsystems are virtualized by intercepting their network requests and redirecting them to Honeyd. Every configuration template may contain subsystems that are started as separated processes when the template is bound to a virtual IP address. An additional benefit

of this approach is the ability of honeypots to create periodic background traffic like requesting web pages and reading email, etc.

There are certain *disadvantages* of honeypots: all network traffic received by a honeypot is considered by definition to be suspicious, as the system has an idle role and its existence is not advertised. Unfortunately, even idle connected systems receive plenty of noise traffic, which makes it harder for administrators to identify malicious from innocuous traffic. To overcome this issue, dynamic analysis systems have been brought into play to host high-interaction honeypots.
Another weakness of honeypots is that by design they support attacks that perform target discovery through network scans. As technologies like IPv6 and network address translation (NAT) become more popular, scanning has become less efficient, and attackers have turned to other means to discover targets. As a response, we have witnessed the development of client-side honeypots, which by continuously connecting to remote servers (mostly web servers admittedly) attempt to discover malicious ones.

Table 1. Top 10 Detections for February 2011 as reported by GFI Software.

Detection	Type	Percent
Trojan.Win32.Generic!BT	Trojan	22.97%
Trojan-Spy.Win32.Zbot.gen	Trojan	3.46%
Trojan.Win32.Generic.pak!cobra	Trojan	2.89%
Zugo LTD (v)	Adware	2.52%
Fraudtool.Win32.Securityshield.ek!c (v)	Trojan	2.00%
Trojan.Win32.Generic!SB.0	Trojan	1.72%
INF.Autorun (v)	Trojan	1.66%
Worm.Win32.Downad.Gen (v)	Worm	1.48%
Pinball Corporation (v)	Adware	1.19%
Exploit.PDF-JS.Gen (v)	PDF exploit	0.83%

3.3 Honeycomb

Honeycomb is realized as a *Honeyd* extension. It is based on the idea that any traffic directed to the honeypot can be considered an attack. Figure 1 shows the high-level overview of honeycomb's signature creation algorithm. *Honeycomb* automatically generates *Snort a*nd *Bro* signatures for all incoming traffic. New signatures are created if a similar pattern does not yet exist. Existing signatures are updated whenever similar traffic has been detected, so the quality of the signatures is increased with each similar attack session. Signatures can be updated to match mutations of existing attacks. For each mutation a more generic description for the signature is generated, so that the original attack and the mutation are both matched. This way the signature base is kept small. The mechanism creates signatures for all traffic directed to the honeypot. Unfortunately the attacks are not verified to be successful in any way. Therefore, it suffers of false positives if any non-attack traffic is directed to the honeypot like e.g. the IPX protocol. A computer connected to the Internet especially on a dial-up connection is addressed even by non attack traffic. Whenever a search engine tries to mirror the host or a peer to peer program tries to connect, a signature is generated. Signatures must be checked manually afterwards whether they were created for an attack or for something else. An approach to verify the attack patterns is desirable. The signature generation mechanism could be used to create IDS signatures if appropriate attack traffic is identified and directed to the system.

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.

4. ALGORITHMS

4.1 Dynamic Taint Analysis

Dynamic taint analysis (DTA) is a mechanism to tracks incoming data from the network throughout the process. The un-trusted data are marked as 'tainted' originating from the network. When operations on this data are performed, taint tags are propagated to the result of such operations. An alert is raised and relevant action is performed when data from a tainted piece of memory is used in an important operation, for example as target address in a jump.

Tainting data
Tainting of data can be done by adding an integrity bit to every 32-bit word of memory. It then can use Biba's low-watermark integrity policy with values \high" and \low" to describe the level of threat the data poses.

Taint propagation
Taint propagation occurs when arithmetic is performed with tainted values, like for example a value x_t is increased with the tainted variable n; the resulting y_t is also tainted.

$$x_t + n = y_t$$

The register is also marked tainted containing tainted memory. Logging of the 'tainted' marks is generally done by adding some memory structures containing the tags for its memory section. Paging techniques can be used for optimization to lower the overhead.

4.2 LCS algorithm

Longest Common Substring is a popular and fast algorithm for detecting patterns between multiple strings used by automatic signature generation projects. The algorithm finds the longest substring that is common to memory and traffic trace. The longer common substring is computed between two packets, for the suspected anomalous similar incoming/outgoing packets. The main disadvantage is the computation overhead. LCS can be implemented in linear time and avoids the fragmentation problem and other small payload manipulations.

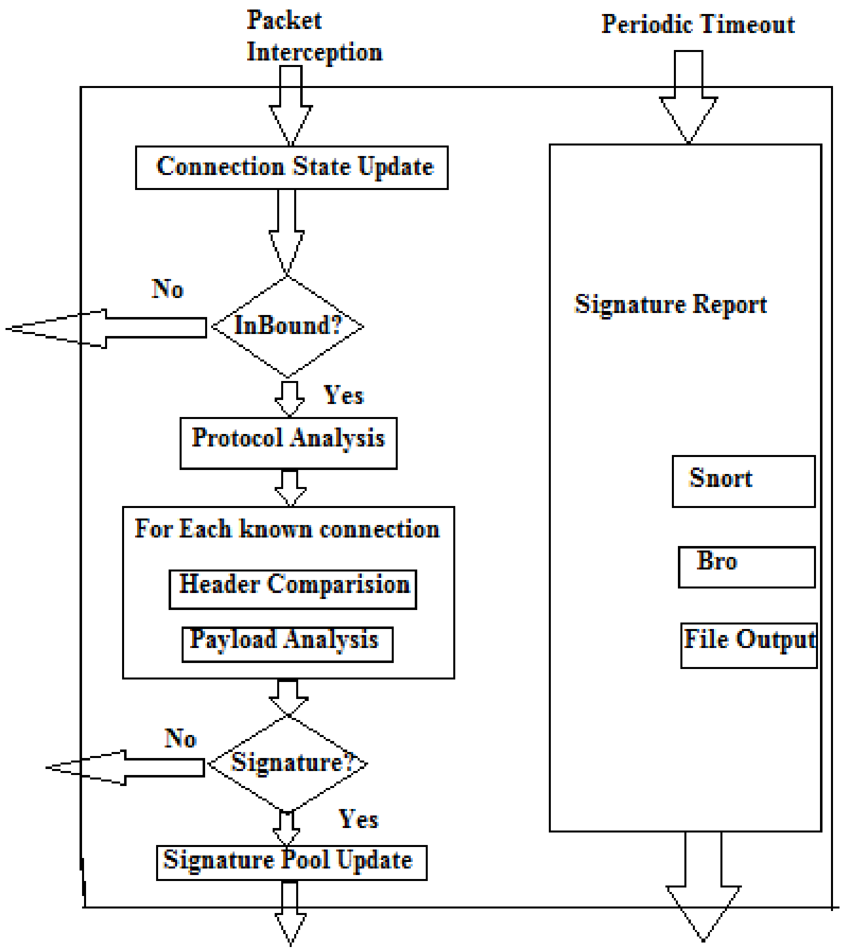

Fig 1: High level overview of Honeycomb's Signature Creation Algorithm

5. CONCLUSION

In this paper we have seen how the problem of worm propagation arises into computer network, which is basically due to software errors that incurred into software binaries during development phase. The self propagating malware do not need any human intervention to propagate into the network. Also, we have discovered how the unknown worm signatures are considered as Zero-Day worm signatures. We have shown the possible defenses such as Intrusion detection system and Honeypots. Honey pots are systems used to attract and fool the hackers by exposing known vulnerabilities by virtualzing the well-known services. The two algorithms, Dynamic Taint Analysis and LCS algorithm are capable to detect the signature of known attacks. *Honeycomb* automatically generates *Snort and Bro* signatures for all incoming traffic and new signatures are created if a similar pattern does not yet exist using LCS algorithm. Finally, Honeycomb which is basically Honeyd extension can be effectively used to detect the unknown worm-"Zero-Day" signatures in the virtualized network.

6. ACKNOWLEDGMENTS

I would like to thank Chirag S. Thaker to provide incessant guidance to my work and giving his valuable suggestions regarding the research work methodology. I also like to thank Hemant Patel to guide me about software required for the given project.

7. REFERENCES

[1] C. Xenakis a, C. Panos b, I. Stavrakakis b: A comparative evaluation of intrusion detection

architectures for mobile ad hoc networks, elsevier , computers & security 30 (2011) 63 -80

[2] D. Dagon, X. Qin, G. Gu, W. Lee, J. Grizzard, J. Levine, and H. Owen. HoneyStat: Local Worm Detection Using Honeypots. In Proceedings of the 7th International Symposium on Recent Advances in Intrusion Detection (RAID), pages 39-58, October 2004.

[3] Dr I. Muttik , McAfee Labs, UK: ZERO-DAY MALWARE ,Virus bulletin conference September 2010.

[4] G. Portokalidis ,A. Slowinska, H. Bos: Argos: an Emulator for Fingerprinting Zero-Day Attacks for advertised honeypots with automatic signature generation, EUROSYS 2006

[5] Honeynet Project. Know Your Enemy: Statistics. http://project.honeynet.org/papers/stats/, July 2001.

[6] Honeynet Project. Know Your Enemy: Worms at War. http://project.honeynet.org/papers/worm/, November 2000.

[7] http://www.computerweekly.com/blogs/read-all-about-it/2011/08/none-of-10-top-malware-vulnera.html.

[8] I. Kim, D. Kim, B. Kim, Y. Choi, S.Yoon, J. Oh and J. Jang:An Architecture of Unknown Attack Detection System against Zero-dayWorm, Proceedings of the 8th WSEAS International Conference on APPLIED COMPUTER SCIENCE (ACS'08)

[9] J.Newsome and D.Dong. Dynamic Taint Analysis for Automatic Detection Analysis, and Signature Generation of Exploits on Commodity software. In Proceedings of the 12th ISOC Symposium on Network and Distributed System Security(SNDSS), pages 221-237, February 2005.

[10] Kreibich, C., Crowcroft,J.: Honeycomb-Creating Intrusion Detection Signatures Using Honeypots. ACM SIGCOMM Computer Communication Review 34(2004).

[11] N. Provos. A virtual honeypot framework. In Proc. of the 13th USENIX Security Symposium, 2004.

[12] P. Laskov, M. Kloft: A Framework for Quantitative Security Analysis of Machine Learning, AISec’09, November 9, 2009, Chicago, Illinois, USA.

[13] S. Pastrana, A.Orfila, A.Ribagorda: A Functional Framework to Evade Network IDS , Proceedings of the 44th Hawaii International Conference on System Sciences - 2011.

[14] S. Singh, C. Estan, G. Varghese and S. Savage. Automated Worm Fingerprinting, Sixth Symposium on Operating Systems Design and Implementation (OSDI), 2004.