



A Survey on Worm Detection Technique

Lokesh. M. R*, Vishwa Lalit, Jeeth Nair, Sura Sannith, Pradeep D

Department Information Science and Engineering, & New Horizon College of Engineering,
Bangalore, India

Abstract— *The Code Red Worm This paper focuses on the behavior of the Code Red (CRv2) and Code Red II worms, discussing how the worms propagated and capitalized on known yet unpatched software vulnerabilities. Creating a computer and network security policy is discussed in light of the lessons learned from these worms. This paper addresses the vulnerability that was present in Microsoft Internet Information Services (IIS) web server software and the worm, Code Red, which exploited this vulnerability. It describes the mechanisms of three different versions of Code Red, as well as the patches and methods for stopping the worm. Finally, it discusses the effects of the worm, financially and technically, as well as in how it has impacted the security of systems in general is an outcome of this paper.*

Keywords— *Code Red (CRv2) worms, Code Red II worms, Slammer worms, CAIDA*

I. INTRODUCTION

It is common for viruses, hacker attacks and system vulnerabilities to make the evening news on an almost weekly basis. Web sites such as SANS and CERT are updated daily with new viruses, worms and security holes. It has become a difficult task for system administrators to keep up with the task of securing their systems. Not only must they know about the constantly changing vulnerabilities that are present in software and hardware, but they must also continue to monitor for attacks, patch for new viruses and control access of internal users.

Exploits can come in many forms. Viruses such as “I Love You” and “Melissa” can affect individual computers and web traffic through launching email attachments with malicious code. Denial of Service attacks send traffic to flood servers and bring them down. But a worm is an exploit that is often times more effective than other methods because it invades servers overwhelming the memory capacity and then shuts down before the worm is passed automatically to another machine. It is also effective because it does not rely on infected files spreading through the “cooperation” of a user opening an email attachment or launching a program. Recently an exploit was identified and named. This worm caused billions of dollars in damages and introduced to the technology community the dangers of not reacting quickly to public warnings of vulnerabilities.

This paper addresses the vulnerability that was present in Microsoft Internet Information Services (IIS) web server software and the worm, which exploited this vulnerability. It describes the mechanisms of three different versions of, as well as the patches and methods for stopping the worm.

Finally, it discusses the effects of the worm, both financially and technically, as well as in how it has impacted the security of systems in general. Various methods and flows given by the experts are discussed in the paper which deals with the problem of User Security. Some of the Solutions with the statistics are given below in the paper. In Section I, we review relevant analysis research on CAIDA Analysis of Code-Red . In Section II we present SQL Slammer Worm that causes a denial of service on some Internet hosts, In Sections III we describe about Worm Containment designed to halt the spread of a worm in an enterprise and finally we conclude by discussing about today's containment techniques Scan Suppression which responds to detected ports scans by blocking future scanning attempts

II. DIFFERENT TYPES OF WORMS DETECTION TECHNIQUE

An easy way to comply with the conference paper formatting requirements is to use this document as a template and simply type your text into it.

A. CAIDA Analysis of Code-Red

The first incarnation of the Code-Red worm (CRv1) began to infect hosts running unpatched versions of Microsoft's IIS webserver on July 12th, 2001. The first version of the worm uses a static seed for it's random number generator. Then, around 10:00 UTC in the morning of July 19th, 2001, a random seed variant of the Code-Red worm (CRv2) appeared and spread. This second version shared almost all of its code with the first version, but spread much more rapidly. Finally, on August 4th, a new worm began to infect machines exploiting the same vulnerability in Microsoft's IIS webserver as the original Code-Red virus. Although the new worm shared almost no code with the two versions of the original worm, it contained in its source code the string "CodeRedII" and was thus named CodeRed II. The characteristics of each worm are explained in greater detail below.

1. The IIS .ida Vulnerability:

Detailed information about the IIS .ida vulnerability can be found at eEye

On June 18, 2001 eEye released information about a buffer-overflow vulnerability in Microsoft's IIS web servers. The remotely exploitable vulnerability was discovered by Riley Hassell. It allows system-level execution of code and thus presents a serious security risk. The buffer-overflow is exploitable because the ISAPI (Internet Server Application Program Interface) .ida (indexing service) filter fails to perform adequate bounds checking on its input buffers.

A security patch for this vulnerability is available on Microsoft.

Code-Red version 1 (CRv1)

On July 12, 2001, a worm began to exploit the aforementioned buffer-overflow vulnerability in Microsoft's IIS web servers. Upon infecting a machine, the worm checks to see if the date (as kept by the system clock) is between the first and the nineteenth of the month. If so, the worm generates a random list of IP addresses and probes each machine on the list in an attempt to infect as many computers as possible. However, this first version of the worm uses a static seed in its random number generator and thus generates identical lists of IP addresses on each infected machine. The first version of the worm spread slowly, because each infected machine began to spread the worm by probing machines that were either infected or impregnable. The worm is programmed to stop infecting other machines on the 20th of every month. In its next attack phase, the worm launches a Denial-of-Service attack against www1.whitehouse.gov from the 20th-28th of each month.

On July 13th, Ryan Perme and Marc Maiffret at eEye Digital Security received logs of attacks by the worm and worked through the night to disassemble and analyze the worm. They christened the worm "Code-Red" both because the highly caffeinated "" Mountain Dew fueled their efforts to understand the workings of the worm and because the worm defaces some web pages with the phrase "Hacked by Chinese". There is no evidence either supporting or refuting the involvement of Chinese hackers with the Code-Red worm.

The first version of the Code-Red worm caused very little damage. The worm did deface web pages on some machines with the phrase "Hacked by Chinese." Although the worm's attempts to spread itself consumed resources on infected machines and local area networks, it had little impact on global resources.

The Code-Red version 1 worm is memory resident, so an infected machine can be disinfected by simply rebooting it. However, once-rebooted, the machine is still vulnerable to repeat infection. Any machines infected by Code-Red version 1 and subsequently rebooted were likely to be re-infected, because each newly infected machine probes the same list of IP addresses in the same order.

Code-Red version 2

At approximately 10:00 UTC in the morning of July 19th, 2001 a random seed variant of the Code-Red worm (CRv2) began to infect hosts running unpatched versions of Microsoft's IIS web server. The worm again spreads by probing random IP addresses and infecting all hosts vulnerable to the IIS exploit. Code-Red version 2 lacks the static seed found in the random number generator of Code-Red version 1. In contrast, Code-Red version 2 uses random seed, so each infected computer tries to infect a different list of randomly generated IP addresses. This seemingly minor change had a major impact: more than 359,000 machines were infected with Code-Red version 2 in just fourteen hours.

Because Code-Red version 2 is identical to Code-Red version 1 in all respects except the seed for its random number generator, its only actual damage is the "Hacked by Chinese" message added to top level webpages on some hosts. However, Code-Red version 2 had a greater impact on global infrastructure due to the sheer volume of hosts infected and probes sent to infect new hosts. Code-Red version 2 also wreaked havoc on some additional devices with web interfaces, such as routers, switches, DSL modems, and printers. Although these devices were not infected with the worm, they either crashed or rebooted when an infected machine attempted to send them a copy of the worm.

Like Code-Red version 1, Code-Red version 2 can be removed from a computer simply by rebooting it. However, rebooting the machine does not prevent reinfection once the machine is online again. On July 19th, the probe rate to hosts was so high that many machines were infected as the patch for the .ida vulnerability was applied.

On August 4, 2001, an entirely new worm, CodeRedII began to exploit the buffer-overflow vulnerability in Microsoft's IIS web servers. Although the new worm is completely unrelated to the original Code-Red worm, the source code of the worm contained the string "CodeRedII" which became the name of the new worm.

Ryan Perme and Marc Maiffret analyzed Code RedII to determine its attack mechanism. When a worm infects a new host, it first determines if the system has already been infected. If not, the worm initiates its propagation mechanism, sets up a "backdoor" into the infected machine, becomes dormant for a day, and then reboots the machine. Unlike Code-Red, Code RedII is not memory resident, so rebooting an infected machine does not eliminate Code RedII.

After rebooting the machine, the Code RedII worm begins to spread. If the host infected with Code RedII has Chinese (Taiwanese) or Chinese (PRC) as the system language, it uses 600 threads to probe other machines. All other machines use 300 threads. CodeRedII uses a more complex method of selecting hosts to probe than Code-Red. CodeRedII generates a random IP address and then applies a mask to produce the IP address to probe. The length of the mask determines the similarity between the IP address of the infected machine and the probed machine. 1/8th of the time, CodeRedII probes a completely random IP address. 1/2 of the time, Code RedII probes a machine in the same /8 (so if the infected machine had the IP address 10.9.8.7, the IP address probed would start with 10.), while 3/8ths of the time, it probes a machine on the same /16 (so the IP address probed would start with 10.9.). Like Code-Red, CodeRedII avoids probing IP addresses in 224.0.0.0/8 (multicast) and 127.0.0.0/8 (loopback). The bias towards the local /16 and /8 networks means that an infected machine may be more likely to probe a susceptible machine, based on the supposition that machines on a single network are more likely to be running the same software as machines on unrelated IP addresses.

The CodeRedII worm is much more dangerous than Code-Red because CodeRedII installs a mechanism for remote, root-level access to the infected machine. Unlike Code-Red, CodeRedII neither defaces web pages on infected machines

nor launches a Denial-of-Service attack. However, the backdoor installed on the machine allows any code to be executed, so the machines could be used as zombies for future attacks (DoS or otherwise).

A machine infected with CodeRedII must be patched to prevent reinfection and then the CodeRedII worm must be removed. A security patch for this vulnerability is available from Microsoft. A tool that disinfects a computer infected with CodeRedII is also available

2. SQL Slammer Worm

The SQL slammer worm is a computer worm that caused a denial of service on some Internet hosts and dramatically slowed down general Internet traffic, starting at 05:30 UTC on January 25, 2003. It spread rapidly, infecting most of its 75,000 victims within ten minutes. So named by Christopher J. Rouland, the CTO of ISS, Slammer was first brought to the attention of the public by Michael Bacarella - see Notes. Although titled "SQL slammer worm", the program did not use the SQL language; it exploited a buffer overflow bug in Microsoft's flagship SQL Server and Desktop Engine database products, for which a patch had been released six months earlier in MS02-039. Other names include W32.SQLExp.Worm, DDOS.SQLP1434.A, the Sapphire Worm, SQL_HEL, W32/SQL Slammer and Helkern.

Sites monitoring the traffic of the Internet such as Internet Storm Center reported significant slowdowns globally, resembling the effects of the worm in the summer of 2001.

Yonhap news agency in South Korea reported that Internet services had been shut down for hours on Saturday, January 25, 2003 nationwide. The effects were mitigated by the fact that it occurred over the weekend.

The same attack was reported throughout most of Asia, Europe, and North America. Anti-virus software maker Symantec estimated that at least 22,000 systems were affected worldwide.

The Microsoft SQL Server Desktop Engine (MSDE) was affected by the worm and that increased the number of the systems affected. This, together with many home users unaware they have MSDE installed, worsened the effects of this worm. Also, if a computer running MSDE was infected with this worm via the Internet and then connected to a Virtual Private Network, the SQL Servers inside the NAT could be infected.

According to a CAIDA-coordinated analysis of the SQL Slammer outbreak, its growth followed an exponential curve with a doubling time of 8.5 seconds in the early phases of the attack, which was only slowed by the collapse of many networks because of the denial of service caused by SQL Slammer's traffic. 90% of all vulnerable machines were infected within 10 minutes, showing that the original estimate for infection speed was roughly correct.

Technical details

The worm was based on proof of concept code demonstrated at the Black Hat Briefings by David Litchfield, who had initially discovered the buffer overflow vulnerability that the worm exploited. It is a small piece of code that does little other than generate random IP addresses and send itself out to those addresses. If a selected address happens to belong to a host that is running an unpatched copy of Microsoft SQL Server Resolution Service, the host immediately becomes infected and begins spraying the Internet with more copies of the worm program.

Home PCs are generally not vulnerable to this worm unless they have MSDE installed. The worm is so small that it does not contain code to write itself to disk, so it only stays in memory, and it is easy to remove. For example, Symantec provides a free removal utility (see external link below), or it can even be removed by restarting SQL Server (although the machine would likely be immediately reinfected).

The worm was made possible by a software security vulnerability in SQL Server first reported by Microsoft on July 24, 2002. A patch had been available from Microsoft for six months prior to the worm's launch, but many installations had not been patched – including some at Microsoft.

The slowdown was caused by the collapse of numerous routers under the burden of extremely high bombardment traffic from infected servers. Normally, when traffic is too high for routers to handle, the routers are supposed to delay or temporarily stop network traffic. Instead, some routers crashed (became unusable), and the "neighbor" routers would notice that these routers had stopped and should not be contacted (aka "removed from the routing table"). Routers started sending notices to this effect to other routers they knew about. The flood of routing table update notices caused some additional routers to fail, compounding the problem. Eventually the crashed routers' maintainers restarted them, causing them to announce their status, leading to another wave of routing table updates. Soon a significant portion of Internet bandwidth was consumed by routers communicating with each other to update their routing tables, and ordinary data traffic slowed down or in some cases stopped altogether. Ironically because the SQL slammer worm was so small in size, sometimes it was able to get through and legitimate traffic was not.

SQL Slammer was the first observed example of a "Warhol worm" – a fast-propagating Internet infection of the sort first hypothesized in 2002 in a paper by Nicholas Weaver. Two key aspects contributed to SQL Slammer's rapid propagation. The worm infected new hosts over UDP, and the entire worm (only 376 bytes) fits inside a single packet. As a result, no connection was necessary for an infected host to attempt to infect another machine. Each infected host could instead simply "fire and forget" packets as rapidly as possible (generally hundreds per second).

How it works

SQL Slammer exploits the way in which MS SQL servers process input on SQL Server Resolution Service port 1434. A specially crafted packet of only 376 bytes sent over the Internet can remotely compromise a vulnerable server. The SQL worm itself is file-less and resides only in memory, much as . It does not create or delete files but actively scans for other vulnerable MS SQL servers. The aggressive scanning done by SQL Slammer overloaded many networks on January 25, 2003, slowing Internet traffic.

SQL Slammer targets systems running MS SQL Server 2000 and/or systems running Microsoft Desktop Engine (MSDE) 2000, which is included in Visual Studio .Net, Asp.net Web Matrix Tool, Office XP Developer Edition, MSDN

Universal and Enterprise, Microsoft Access, and Microsoft Application Center 2000.

3. Worm Containment

Worm containment is designed to halt the spread of a worm in an enterprise by detecting infected machines and preventing them from contacting further systems. Current approaches to containment are based on detecting the scanning activity associated with scanning Phishing, as is our new algorithm.

Scanning Phishing operate by picking "random" addresses and attempting to infect them. The actual selection technique can vary considerably, from linear scanning of an address space (Blaster), fully random (), a bias toward local addresses (II] and Nimda), or even more enhanced techniques (Permutation Scanning). While future Phishing could alter their style of scanning to try to avoid detection, all scanning Phishing share two common properties: most scanning attempts result in failure, and infected machines will institute many connection attempts. Because containment looks for a class of behavior rather than specific worm signatures, such systems can stop new (scanning) Phishing.

Robust worm defense requires an approach like containment because we know from experience that Phishing can find (by brute force) small holes in firewalls, VPN tunnels from other institutions, infected notebook computers, web browser vulnerabilities, and email-borne attacks to establish a foothold in a target institution. Many institutions with solid firewalls have still succumbed to Phishing that entered through such means. Without containment, even a single breach can lead to a complete internal infection.

Along with the epidemic threshold and sustained sub-threshold scanning, a significant issue with containment is the need for complete deployment within an enterprise. Otherwise, any uncontained-but-infected machines will be able to scan through the enterprise and infect other systems. (A single machine, scanning at only 10 IP addresses per second, can scan through an entire in under 2 hours.)

Thus, we strongly believe that worm-suppression needs to be built into the network fabric. When a worm compromises a machine, the worm can defeat host software designed to limit the infection; indeed, it is already common practice for viruses and mail-Phishing to disable antivirus software, so we must assume that future Phishing will disable worm-suppression software.

Additionally, since containment works best when the cells are small, this strongly suggests that worm containment needs to be integrated into the network's outer switches or similar hardware elements, as proximate to the end hosts as economically feasible. This becomes even more important for cooperative containment, as this mechanism is based on some cells becoming compromised as a means of better detecting the spread of a worm and calibrating the response necessary to stop it.

4. Scan Suppression

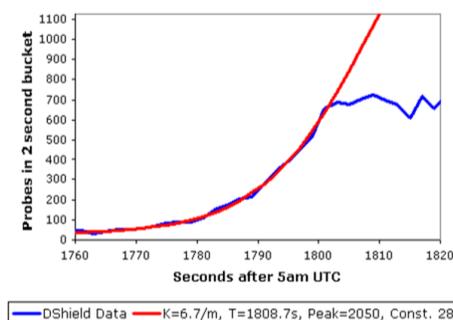
The key component for today's containment techniques is scan suppression: responding to detected ports scans by blocking future scanning attempts. Port scans--probe attempts to determine if a service is operating at a target IP address--are used by both human attackers and Phishing to discover new victims. Port scans have two basic types: horizontal scans, which search for an identical service on a large number of machines, and vertical scans, which examine an individual machine to discover all running services. (Clearly, an attacker can also combine these and scan many services on many machines. For ease of exposition, though, we will consider the two types separately.)

The goal of scan suppression is often expressed in terms of preventing scans coming from "outside" inbound to the "inside." If "outside" is defined as the external Internet, scan suppression can thwart naive attackers. But it can't prevent infection from external Phishing because during the early portion of a worm outbreak an inbound-scan detector may only observe a few (perhaps only single) scans from any individual source. Thus, unless the suppression device halts all new activity on the target port (potentially disastrous in terms of collateral damage), it will be unable to decide, based on a single request from a previously unseen source, whether that request is benign or an infection attempt.

For worm containment, however, we turn the scan suppressor around: "inside" becomes the enterprise's larger internal network, to be protected from the "outside" local area network. Now any scanning worm will be quickly detected and stopped, because (nearly) all of the infectee's traffic will be seen by the detector.

We derived our scan detection algorithm from TRW (Threshold Random Walk) scan detection. In abstract terms, the algorithm operates by using an oracle to determine if a connection will fail or succeed. A successfully completed connection drives a random walk upwards; a failure to connect drives it downwards. By modeling the benign traffic as having a different (higher) probability of success than attack traffic, TRW can then make a decision regarding the likelihood that a particular series of connection attempts from a given host reflect benign or attack activity, based on how far the random walk deviates above or below the origin. By casting the problem in a Bayesian random walk framework,

DSshield Probe Data



TRW can provide deviation thresholds that correspond to specific false positive and false negative rates, if we can parameterize it with good a priori probabilities for the rate of benign and attacker connection successes.

- Infection stats:
 - i. 90% in 10 minutes
 - ii. pop doubled every 8.5s
 - iii. >=75000 infected
 - iv. 1 UDP packet!

To implement TRW, we obviously can't rely on having a connection oracle handy, but must instead track connection establishment. Furthermore, we must do so using data structures amenable to high-speed hardware implementation, which constrains us considerably. Finally, TRW has one added degree of complexity not mentioned above. It only considers the success or failure of connection attempts to new addresses. If a source repeatedly contacts the same host, TRW does its random walk accounting and decision-making only for the first attempt. This approach inevitably requires a very large amount of state to keep track of which pairs of addresses have already tried to connect, too costly for our goal of a line-rate hardware implementation. As developed in Section 5, our technique uses a number of approximations of TRW's exact bookkeeping, yet still achieves quite good results.

There are two significant alternate scan detection mechanisms proposed for worm containment. The first is the new-destination metric proposed by Williamson. This measures the number of new destinations a host can visit in a given period of time, usually set to 1 per second. The second is dark-address detection, used by both Forescout and Mirage Networks. In these detectors, the device routes or knows some otherwise unoccupied address spaces within the internal network and detects when systems attempt to contact these unused addresses.

III. CONCLUSIONS

Phishing has becoming a serious network security problem, causing financial loss of billions of dollars to both consumers and e-commerce companies. And perhaps more fundamentally, phishing has made e-commerce distrusted and less attractive to normal consumers. In this research paper, we have studied the characteristics of the hyperlinks that were embedded in phishing e-mails. We then designed an anti-phishing algorithm, Link-Guard, based on the derived characteristics. Since Phishing-Guard is characteristic based, it can not only detect known attacks, but also is effective to the unknown ones. We have implemented LinkGuard for Windows XP. Our experiment showed that LinkGuard is light-weighted and can detect upto 96% unknown phishing attacks in real-time. We believe that LinkGuard is not only useful for detecting phishing attacks, but also can shield users from malicious or unsolicited links in Web pages and Instant messages. Our future work includes further extending the LinkGuard algorithm, so that it can handle CSS (cross site scripting).

ACKNOWLEDGMENT

The satisfaction and euphoria that accompany the successful completion of any task would be, but impossible without the mention of the people who made it possible, whose constant guidance and encouragement crowned our efforts with success. We also record here the constant encouragement and facilities extended to us by our parents Mr. Prem Kumar Singh, Dr. Anita Rani Singh, Mr. V R Achuthan, Mrs. Geeta, Mr. Devarajan, Mrs. Chella, Mr. Sura Mala Konaiah, Mrs. Sura Rama Finally a note of thanks to the teaching and non-teaching staff of Information Science and Engineering Department for their cooperation extended to us and our friends, who helped me directly or indirectly in the successful completion paper.

REFERENCES

- [1] D. Seeley, —A tour of the worm,|| in Proc. Winter USENIX Conf., Jan. 1989, pp. 287–304. [2]. D. Moore, C. Shannon, and J. Brown, —Code-Red: A case study on the spread and victims of an Internet worm,|| in Proc. 2nd ACM SIGCOMM Workshop on Internet Measurement, Nov. 2002, pp. 273–284.
- [2] CERT/CC Advisories. [Online]. Available: <http://www.cert.org/advisories/>
- [3] Cooperative Association for Internet Data Analysis (CAIDA). [Online].
- [4] Available: <http://www.caida.org>
- [5] SANS Inst. [Online]. Available: <http://www.sans.org>
- [6] D. Verton. DHS Launches Cybersecurity Monitoring Project. [Online]. Available: <http://www.pcworld.com/news/article/0,aid,112764,00.asp>
- [7] D. Denning, —An intrusion detection model,|| IEEE Trans. Software Eng., vol. SE-13, no. 2, pp. 222–232, Feb. 1987.
- [8] D. Anderson, T. Frivold, and A. Valdes, —Next-Generation Intrusion Detection Expert System (Nides): A Summary,|| SRI International, Tech. Rep. SRI-CSL-95-07, May 1995.
- [9] A. Valdes and K. Skinner, —Adaptive, model-based monitoring for cyber attack detection,|| in Proc. 3th Int. Symp. Recent Advances in Intrusion Detection (RAID), Oct. 2000, pp. 80–92. [11]. S. A. Hofmeyr, S. Forrest, and A. Somayaji, —Intrusion detection using sequences of system calls,|| J. Comput. Security, vol. 6, no. 3, pp. 151–180, 1998.
- [10] W. Lee and S. Stolfo, —A framework for constructing features and models for intrusion detection systems,|| ACM Trans. Inf. Syst. Security, vol. 3, no. 4, pp. 227–261, Nov. 2000