Intrusion Detection for Viruses and Worms

Thomas M. Chen
Dept. of Electrical Engineering
Southern Methodist University
PO Box 750338
Dallas, TX 75275-0338
Tel: 214-768-8541
Fax: 214-768-3573
Email: tchen@engr.smu.edu
Web: www.engr.smu.edu/~tchen

1. Introduction

The global economic impact of computer *viruses* and *worms* soars into the range of billions of dollars every year according to reports by Computer Economics. Since early1999 when the Melissa macro virus began the trend of spreading quickly via mass e-mailing, viruses and worms have become a common and persistent problem for all computer users. In the 2003 CSI/FBI Computer Crime and Security Survey, 82 percent of the surveyed organizations reported being effected by viruses and worms [1]. The average reported loss for each organization due to viruses and worms was approximately $200,000. These statistics quantify what every PC owner can testify from personal experience, constantly dealing with virus infections, antivirus software, and operating system patch updates.

Viruses and worms are created for the purpose of spreading to as many computers as possible by self-replication. They differ only in their method of self-replication. Viruses replicate themselves by attaching their program instructions to an ordinary program or document, such that the virus instructions are executed during the execution of the infected program. As conceived by Fred Cohen as a USC graduate student in 1983, virus instructions contain at least two subroutines [2]. The first subroutine carries out the infection by seeking out normal programs to infect and attaching or overwriting a copy of the virus instructions to those programs or files. The second subroutine of the virus carries the "payload" that dictates the actions to be executed on the infected host. In theory, the payload could carry out virtually any action, such as deletion of files, damage to a hard drive, installation of unwanted software (denial of service agents or keyloggers), or installation of a backdoor allowing remote access.

Worms are closely related to viruses in their ability to self-replicate but do not need to attach parasitically to other programs. Worms are stand-alone automated programs designed to exploit a network to seek out, compromise, and infect vulnerable computers with a copy of themselves. Worms are inherently dependent on a

network and seem to have existed since the early days of the ARPANET (the precursor to today's Internet). The term "worm" was created by researchers at Xerox PARC in 1979, who wrote worms to traverse their internal Ethernet LAN seeking idle processors for distributed computing [3]. The most famous early incident was the Morris worm released by Robert Morris Jr., a student at Cornell University, in November 1988. The Morris worm disabled 6,000 computers (10 percent of the ARPANET at that time) in a few hours.

Worms have become much more prevalent as Internet connectivity has become ubiquitous in the past few years. The Internet has created a fertile environment for worms by making it easier for them to move among computers. Ubiquitous Internet connectivity increases the risk of every computer to exposure to malicious mobile code, which has been demonstrated by recent epidemics. In the single week of August 12, 2003, the Internet saw three major worm epidemics one after another. The first was Blaster (or LovSan) targeted to a Windows DCOM RPC (distributed component object model remote procedure call) vulnerability on Windows XP and Windows 2000 PCs [4]. Although Blaster reportedly infected about 400,000 systems, experts reported that the vulnerable population was much larger but the worm did not achieve near its potential spreading rate due to novice programming. Six days later on August 18, 2003, the apparently well-intended Welchia or Nachi worm spread by exploiting the same RPC DCOM vulnerability as Blaster. Although it appeared to be well intended, trying to remove Blaster from infected computers by downloading a security patch from a Microsoft Web site, its probing resulted in serious congestion on some networks, such as Air Canada's check-in system and the US Navy and Marine Corps computers. On the very next day, the very fast Sobig.F worm spread among Windows PCs by e-mail [5]. It worked well because it grabbed e-mail addresses from a variety of different types of files on infected computers and secretly e-mailed itself to all of them, pretending to be sent from one of the addresses. At its peak, Sobig.F accounted for one in every 17 messages, and reportedly produced over 1 million copies of itself within the first 24 hours.

The communal nature of the Internet means that every computer is exposed to new viruses and worms. Although individual computers might avoid infection through antivirus software, experience has shown that a substantial number of computers will likely be inadequately protected and could allow new outbreaks to gain a foothold.. In that case, even uninfected computers will be effected by the probing traffic or other side effects caused by a new outbreak. Therefore, it is in everyone's mutual interest to contain or "quarantine" new outbreaks. The first step to containment is automated detection of a new virus or worm, which falls within the problem of *intrusion detection* [6].

2. Intrusion Detection

Intrusion detection systems (IDSs) are designed to monitor activities on a network and recognize anomalies that may be symptomatic of misuse or malicious attacks. Typically, recognition is based on pattern matching and statistical techniques. The idea of statistical analysis to recognize unusual behavior can be traced back to James Anderson in 1980 [7] and an early IDS prototype called Intrusion Detection Expert System (IDES) sponsored by the U.S. Navy in the mid-1980s [8]. An early experiment to extend intrusion detection beyond operating system audit information to include network traffic data was the Network System Monitor (NSM) at the University of California at Davis [heber]. Various IDS prototypes were researched in the 1980s-1990s, and commercial IDS products appeared in the late 1990s.

As shown in Figure 1, an IDS consists of three components: monitoring, analysis, and response. Data is collected by monitoring activities in the hosts or network. The raw data is analyzed to classify activities as normal or suspicious. When a suspicious activity is considered sufficiently serious, a response is triggered. In most IDSs, the response is simply an alarm sent to the network administrator for further action. A recent trend is to coordinate the alarm with an automatic defensive response, such as blocking network traffic. This approach is motivated by the desire for faster response to attacks. However, triggering automatic defenses could be problematic if the intrusion detection is not reliable or accurate.

Intrusion detection approaches can be classified according to the monitoring location as *host-based* or *network-based* (or a combination of both). Host-based IDS focus on monitoring activities on computers, usually including operating system audit trails and system logs. Monitoring a host can yield valuable information for intrusion detection, but the data might become unreliable if the host becomes compromised. In addition, host-based IDS does not scale very well because the IDS must be running - and expending processing resources - on every host. In contrast, network-based IDS observes traffic on a network usually by means of a packet sniffer on a shared local area network or a switch/router with a mirrored port in a switched network. Network-based IDS is able to watch a number of hosts simultaneously and has the potential to catch an attack at an earlier point than host-based IDS (before it reaches the hosts). Some commercial IDSs combine host-based and network-based techniques.

In addition to the monitoring location, intrusion detection techniques can be classified according to the approach for data analysis to recognize anomalies. The two basic approaches to data analysis are *misuse detection*

and *anomaly detection* (which can be combined). Misuse detection defines a set of attack "signatures" and looks for behavior that matches one of the signatures (hence this approach is sometimes called signature-based IDS). This approach is commonly used in commercial IDS products. Although the concept sounds simple, the approach involves more than simple pattern matching; the most capable analysis engines are able to understand the full protocol stack and perform stateful monitoring of communication sessions. However, this approach is obviously dependent on the accuracy of the signatures. If the signatures are too narrowly defined, some attacks might not be detected; these cases of failed detection are *false negatives*. On the other hand, if signatures are too broadly defined, some benign behavior might be mistaken for suspicious; these false alarms are *false positives*. Signatures should be defined to minimize both false negatives and false positives. In any case, misuse detection would not be able to detect new attacks that do not match a known signature. These failures increase the rate of false negatives. Since new attacks are constantly being discovered, misuse detection has the risk of becoming outdated unless signatures are updated frequently.

In contrast, the anomaly detection approach (sometimes called behavior-based IDS) defines a statistical pattern for "normal" behavior, and any deviations from the pattern are interpreted as suspicious. This approach has two major drawbacks. First, it has been common experience that an accurate definition of normal behavior is a difficult problem. Second, all deviations from normal behavior is classified as suspicious, but only a small fraction of suspicious cases may truly represent an attack. Thus anomaly detection could result in a high rate of false positives if every suspicious case raised an alarm. To reduce the number of false alarms, additional analysis would be needed to identify the occurrences of actual attacks. The main advantage of this approach is the potential to detect new attacks without a known signature.

3. Intrusion Detection for Viruses/Worms

Intrusion detection for viruses and worms must be automated because the rate of spreading can be much faster than humans can possibly react. By the time that a network administrator reacts to an alarm and analyzes IDS data to detect a worm, it may well be too late to prevent widespread infection. Recent worms such as Slammer/Sapphire or Blaster were able to spread to millions of computers within minutes. Automated intrusion detection should attempt to catch a new virus/worm outbreak as early as possible because epidemiology has shown

that new outbreaks spread initially at an exponential rate. Therefore, early detection can have a critical effect on slowing down an outbreak.

At first thought, host-based IDS may seem to be the most logical approach because antivirus software is designed to detect viruses and worms. It would be easy to envision a host-based IDS with desktop antivirus software running on every host which could broadcast an alarm to neighbors if a virus is found. Although this idea is attractive, there are serious drawbacks. First, there is the cost of installing the IDS on every host. Second, experience has shown that antivirus software is often not installed, turned on, or updated, for various reasons. Third, viruses and worms are capable of impairing or turning off antivirus programs. Hence, hosts cannot be considered to be completely reliable for intrusion detection. Lastly, host-based IDS can detect a virus only after it has already reached a host, when it might be too late.

Network-based IDS is often implemented as a firewall integrated with antivirus software. Firewalls offer a natural point for virus/worm scanning because they are typically located at the entry point into a network. In addition, firewalls are designed specifically with capabilities to inspect and filter traffic at high link speeds. Firewalls are not the only possibilities; antivirus scanning can also be integrated into gateways, routers, or switches. In contrast to host-based IDS, network-based IDS has the potential to catch a new virus or worm at an earlier point in its attack, before it reaches the hosts.

Network-based IDS may not have the disadvantages of host-based IDS, but has its own challenges. First, firewalls are not perfect filters and can be circumvented. Firewalls are well suited for blocking traffic directed to specific ports known to be dangerous, for example, but exploits may still be carried out through allowed ports (such as port 80 HTTP). Second, firewalls are effective when configured with specific signatures to look for, but worms and viruses do not have a common signature. Each one can spread in different ways, taking advantage of any number of known vulnerabilities. It is also possible that a new worm might exploit an unknown vulnerability. Third, although a vulnerability may be known, a worm exploiting that vulnerability might appear so soon after discovery of the vulnerability that there is not enough time to develop and distribute a signature for the attack.

The problem of detecting new viruses/worms can be better appreciated by examination of a few recent examples.

- *Code Red worm:* Actually, at least three versions of the Code Red worm have attempted to exploit a buffer overflow vulnerability in Microsoft's IIS web servers, referred to as the Index Server ISAPI

vulnerability, which was announced on June 18, 2001. A buffer overflow attack involves a packet sent to the victim host that exceeds the receiving buffer at the victim in a certain way that causes an overflow onto another portion of the victim's CPU execution stack. If the packet is carefully crafted, the buffer overflow could be used to run arbitrary code on the victim host. The first Code Red I (or .ida Code Red) worm appeared on July 12, 2001 about a month after the discovery of the vulnerability [9]. Upon infection, the worm generated a pseudorandom list of IP addresses to probe, but an apparent mistake in the programming resulted in identical lists of IP addresses generated on each infected host. As a result, the spread was slow because each copy of the worm probed the same machines. A week later on July 19, a second version of Code Red I (Code Red v2) with the programming error fixed was able to spread much faster because the randomly generated IP addresses to probe were truly random. Code Red v2 worm was able to infect more than 359,000 machines within 14 hours. On August 4, 2000, a new worm called Code Red II was observed exploiting the same buffer overflow vulnerability in IIS web servers. This version did not generate completely random IP addresses to probe. About 1 out of 8 addresses were completely random; 4 out of 8 addresses were within the same class A range of the infected host's address; and 3 out of 8 addresses were within the same class B range of the infected host's address.

- *Nimda worm*: On September 18, 2001, the Nimda worm used a combination of 5 different ways to spread to 450,000 hosts within the first 12 hours [10]. First, it sent itself as an e-mail attachment to addresses from the computer's web cache and default MAPI mailbox. Second, it infected Microsoft IIS web servers, selected at random, through a buffer overflow attack called a "unicode Web traversal exploit" (known for a year prior). Third, it copied itself across open network shares. Fourth, it added Javascript to web pages to infect any web browsers. Finally, it looked for backdoors left by previous Code Red II and Sadmind worms.
- *SQL Slammer/Sapphire worm*: Slammer/Sapphire appeared on January 25, 2003, exploiting a buffer overflow vulnerability in Microsoft SQL Server announced six months prior in July 2002 [11]. It is much simpler than previous worms and fits in a single 404-byte UDP packet. In contrast, Code Red was about 4,000 bytes and Nimda was 60,000 bytes. Upon infection, the worm simply generates UDP packets carrying copies of itself at the maximum rate of the computer. The spreading rate was surprisingly fast, reportedly infecting 90 percent of vulnerable hosts within 10 minutes (about 120,000 servers).

- *Blaster worm*: Blaster appeared on August 12, 2003, targeted to a Windows DCOM RPC (distributed component object model remote procedure call) vulnerability announced only a month earlier on July 16, 2003 [4]. The worm probes for a DCOM interface with RPC listening on TCP port 135 on Windows XP and Windows 2000 PCs. Through a buffer overflow attack, the worm causes the target machine to start a remote shell on port 4444 and send a notification to the attacking machine on UDP port 69. A tftp (trival file transfer protocol) "get" command is then sent to port 4444, causing the target machine to fetch a copy of the worm as the file "msblast.exe."

These examples show that worms can have different ways of spreading, and therefore detection of them will be different. For example, Code Red can be detected by first an increase in TCP port 80 scans (when the worm tries to find IIS web servers) and then an increase in crafted HTTP "get" requests looking for an ".ida" file (a buffer overflow attack). The Nimda worm is suggested by an increase in attempts of the unicode Web traversal exploit. Slammer/Sapphire is evident from sudden congestion on links and an increase in 404-byte UDP packets directed to UDP port 1434 at random addresses. Blaster will increase the number of TCP packets directed to TCP port 135 and the number of tftp requests on UDP port 69.

These signs of worms are much more obvious in hindsight, after a worm has been thoroughly studied. The problem is that a new worm attack might exploit any number of many known vulnerabilities. A more serious possibility that a new worm might take advantage of a new unknown vulnerability. Signature-based detection would obviously fail for new attacks which do not match a known signature. Anomaly detection looking for "worm-like" behavior is more promising for detecting new unknown attacks. However, anomaly detection is difficult because worms do not share a single typical behavior. Worms might exhibit certain signs, such as: a dramatic increase in the volume of network traffic, perhaps congestion; a steady increase in scans and probes (especially a multiplication in a specific type of scan); and a sudden change in the traffic behavior of hosts (symptomatic of an infected host). However, these signs are not completely reliable indicators of a worm attack. For example, port scans are always going on in the background traffic of the Internet. Sudden congestion may be caused by a number of ordinary reasons. It is difficult to identify a worm attack for certain until a worm has been captured.

4. Open Issues

The main problem with intrusion detection is its accuracy. The goal of intrusion detection is 100 percent detection accuracy with no false positives or false negatives, but current IDS technology is not close to achieving that level of accuracy or reliability. It is particularly difficult to reliably detect new attacks. The majority of IDS products depend on misuse detection but this is limited to known signatures and depend on constant updates. Many IDS products combine misuse detection and anomaly detection because anomaly detection is is more promising to detect new unknown attacks. The challenge in anomaly detection is to minimize false positives.

Performing intrusion detection in real time is an important requirement for virus/worm attacks because new outbreaks must be contained as early as possible. Without real-time detection, viruses and worms might be able to outrace security defenses. However, high-speed networks will involve enormous volumes of data to collect and process rapidly. The capabilities for real-time traffic processing is built into modern firewalls and routers/switches, but the fraction of traffic that is actual attack traffic might be very small. That is, enormous processing power is spent to detect rare events.

Finally, a recent trend is to combine intrusion detection with an active response system, such as automatic configuration of routers or firewalls to block attack traffic. Traditionally, IDSs simply generate alarms to the network administrator to take further action. Typically system administrators are required to sift through voluminous logs of alerts to identify real intrusions. This process would be too slow and time consuming for viruses and worms, which can spread within minutes. Automated response is important to contain a virus/worm outbreak at an early stage. Unfortunately, the unreliability and inaccuracy of current IDSs is a problem. An automated response to false alarms could be the completely wrong course of action. Again, the accuracy of intrusion detection is the key issue.

References

[1] R. Richardson, "2003 CSI/FBI Computer crime and security survey," available at http://www.goscsi.com.

[2] F. Cohen, "Computer viruses: theory and experiments," *Computers and Security*, vol. 6, pp. 22-35, Feb. 1987.

[3] J. Shoch, J. Hupp, "The 'worm' programs - early experience with a distributed computation," *Communications of ACM*, vol. 25 , pp. 172-180, March 1982.

[4] CERT advisory CA-2003-20, "W32/Blaster worm," available at http://www.cert.org/advisories/CA-2003-20.html.

[5] Symantec Security Response, "W32.Sobig.F@mm," available at http://securityresponse.symantec.com/avcenter/venc/data/w32.sobig.f@mm.html.

[6] S. Northcutt, J. Novak, *Network Intrusion Detection*, 3rd ed., Pearson Education, 2003.

[7] J. Anderson, *Computer Security Threat Monitoring and Surveillance*, James P. Anderson Co., Fort Washington, PA, 1980.

[8] D. Denning, "An intrusion detection model," *IEEE Transactions on Software Engineering*, vol. 13, pp. 222-232, Feb. 1987.

[9] CERT incident note IN-2001-08, “Code Red worm exploiting buffer overflow in IIS indexing service DLL,” available at http://www.cert.org/incident_notes/IN-2001-08.html.
[10] CERT advisory CA-2001-26, “Nimda worm,” available at http://www.cert.org/advisories/CA-2001-26.html.
[11] CERT advisory CA-2003-04, “MS-SQL server worm,” available at http://www.cert.org/advisories/CA-2003-04.html.