International Journal of Scientific and Innovative Research 2014; 2(1): 115-122,
P-ISSN 2347-2189, E- ISSN 2347-4971

A STUDY OF VARIOUS WORMS AND THEIR DETECTION SCHEMES

***SUCHETA[1], K P YADAV[2]**
[1]Research Scholar, Department of Computer Science, Sai Nath University, Ranchi, India
[2] Director, MIET, Greater Noida, Uttar Pradesh, India

*Address for correspondence: Sucheta, Research Scholar,
Department of Computer Science,
Sai Nath University, Ranchi, India

ABSTRACT

Computer worms have a behavior of self-propagation over the host machines and have been terrorizing the Internet for the last several years. This is due to the ability of worms to propagate in automated fashion as they continuously compromise computer on the internet. At the same time, being fully automated, it makes their behavior repetitious and predictable. This article presents a survey on the behavior and detection schemes of Internet worms. We first identify worm characteristics through their behavior, and then classify worm detection algorithms based on the parameters used in the algorithms. Furthermore, we analyze and compare different detection algorithms with reference to the worm characteristics by identifying the type of worms that can and cannot be detected by these schemes.

Keywords: *Self-Propagation, Behavior, Detection, Vulnerability, Algorithms*

INTRODUCTION

Self-propagating malicious codes known as computer worms spread themselves without any human interaction and launch the most destructive attacks against computer networks like launching massive Distributed Denial-of-Service (DDoS) attacks that disrupt the Internet utilities, access confidential information that can be misused through large-scale traffic sniffing, key logging etc. They destroy data that has a high monetary value, and distribute large-scale unsolicited advertisement emails (as spam) or software (as malware). These worms include Camouflaging worm (C-Worm in short)[2], Code-Red worm[3], Slammer worm[4], Witty/Sasser worms[8] and Morris Worm[6]. Being fully automated, a worm's behavior is usually repetitious and predictable, making it possible to be detected. A worm's life consists of the following phases: target finding, transferring, activation, and infection. Since worms involve network activities in the first two phases, their behaviors in these two phases are critical for developing detection algorithms. Therefore, this paper first focuses on worm characteristics that facilitate their detection. Many algorithms have been proposed in the past to try to catch and stop the spread of Internet worms. Most research papers discuss efforts that are related to their proposed work, but none of these papers gives a comprehensive classification of the existing modeling and detection schemes. This paper contains a complete study of some active worms with their behavior and identified various modeling and detection schemes.

Overview

After an introductory terminology is presented, worm characteristics during target finding and worm transferring phases are identified. This is followed by an overview of worm defense mechanisms: modeling and detection. The modeling of various worms and detection schemes are presented next. Depending on where the detection is implemented, they may construct different views of worm propagation behaviors, so there may be differences in the scope of their defenses.

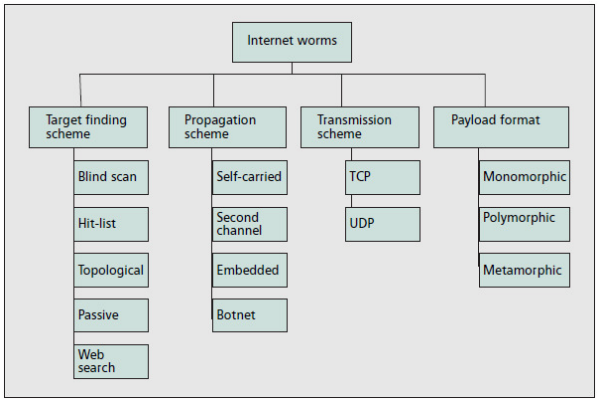

Fig 1: Categorization of worm characteristics

TECHNOLOGY

A. Activation

Activation is when a worm starts performing its malicious activities. Activation might be triggered on a specific date or under certain conditions.

B. False alarm

A false alarm is an incorrect alert generated by a worm detection system.

C. False Positive

A false positive is a false alarm where an alert is generated when there is no actual attack or threat.

D. False negative

False negative means the detection system missed an attack. It is false negative if no alert is generated while the system is under an attack.

E. Infection

Infection is the result of the worm performing its malicious activities on the host.

F. Target Finding

Target finding is the first step in the worm's life to discover the victims (vulnerable hosts).

G. Threshold

Threshold is a predefined condition that, if met, indicates the existence of specious traffic or a worm attack.

H. Transfer

Transfer refers to sending a copy of the worm to the target after the victim (target) is defined.

I. Virus

A virus is a malicious piece of code that attaches to other programs to propagate. It cannot propagate by itself, and normally depends on a certain user intervention, such as opening up an email attachment or running an executable file, to be activated.

J. Worm

A worm is a malicious piece of code that self-propagates, often via network connections, exploiting security flaws in computers on the network. In general, worms do not need any human intervention to propagate; however, a category of worms called passive worms require certain host behavior or human intervention to propagate. For example, a passive worm only propagates itself until it is contacted by another host.

EXISTING INTERNETWORMS

In this section we look at one of the first Internet worms, the Morris worm, which gained extensive media coverage, then discuss five more recent Internet worms: Morris, Code Red, Slammer, and Witty based on their characteristics.

1. Morris Worm

The Morris worm was one of the first Internet worms whose devastating effect gained the wide attention of the media. Morris worm was launched in November 1988 by Robert Tappan Morris, who was a student at Cornell University at the time. It is the first known worm to exploit the buffer overflow vulnerability. It targeted sent mails and finger services on DEC VAX and Sun 3 hosts. Based on the creator's claim, the Morris worm was not intended to cause any harm, but was designed to discover the number of the hosts on the Internet. The worm was supposed to run a process on each infected host to respond to a query if the host was infected by the Morris worm or not. If the answer was yes, the infected host should have been skipped; otherwise, the worm would copy itself to the host. However, a flaw in the program caused the code to copy itself multiple times to already infected machines, each time running a new process, slowing down the infected hosts to the point that they became unusable.

• ***Behaviour***

The Morris worm was a mixture of sophistication and naivety. It had a simple overall design: look at a computer's system configuration to find potential neighbors, invade them, and try to minimize the number of invasions on any machine. The worm used heuristic knowledge about Internet topology and trust relationships to aid its spread, and it targeted two different machine architectures. Its cleverness in finding potential attack targets made it especially effective, but it also took on the time consuming task of guessing passwords on individual user accounts, which gave it an "attack in depth" aspect. Nonetheless, it became a victim of its own success as it was unable to control its exponential growth. With no global information and no point of control, the Morris worm ran rampant.

- It attacked one operating system, but two different computer architectures.
- It had three distinct propagation vectors.
- It had several mechanisms for finding both potential nodes to infect, particularly information about the local system's IP connectivity (its network class and gateway), and information found in user accounts.
- It traversed trusted accounts using password guessing.
- It installed its software via a two-step "hook and haul" method that required the use of a C compiler, link loader, and a call back network connection to the infecting system.
- It evaded notice by obscuring the process parameters and rarely leaving files behind.
- It attempted to limit the re- infection rate on each node (but not the total number).
- It attempted to run forever on as many nodes as possible.

• ***Detection Scheme***

The Morris worm also contained some note -worthy design flaws:

- It was overly aggressive. Although it did have a way to notice multiple infections, its rate-limiting behavior was not effective, and a hundred or more copies could be running on a single machine. Even uninfected machines were vulnerable to assault through multiple infection attempts coming from many independent sources.
- As the number of infections increased, the worm's ability to limit itself decreased. Race conditions in its detection method actually caused the infection rate to increase.
- It could not trace its progress or control it.
- Log files, particularly send mail logs, contained information about the worm's usage. Some log files filled up with the information, and the machines' I/O load increased.
- Its infection method depended on the C compiler, thus preventing access to some major sites that had already established machines that acted as bastions, limiting network access. These machines might not have had C compilers.
- A variety of resource failures left many copies of the "hook" program on the attacked machines.

The intensity of the attacks on machines running the SMTP protocol, particularly those running the Unix send mail program, resulted in denial of email service to large portions of the Internet. This was the Morris worm's most disruptive aspect. Like many human infections, it was not the worm itself that was harmful, but its secondary effects on resources.

2. Code-Red Worm

Code Red I was first seen in July 2001 affecting computers running Microsoft's Internet Information Server (IIS) Web service. In the first 20–25 days after getting into the machine, Code Red I uses a blind scan scheme that scans port 80 on random IP addresses to find other vulnerable machines, and then it launches a denial-of-service (DoS) attack targeting a set of IP addresses. The infected websites will display: "HELLO! Welcome to http://www.worm.com! Hacked By Chinese!" Code Red II was released one month later. It is a variant of the original Code Red. Code Red II no longer launches a

DoS attack against predefined IP addresses; instead, it installs a backdoor into the infected systems. It still employs blind scan but focuses more on the local subnet, and targets mainly systems with Chinese language settings. Code Red I sends its payload in monomorphic format and has a signature starting with "GET /default .ida?NNNNNNN." Code Red II has a similar signature, but replaces N with X. Both versions of Code Red are self-carried and transfer via TCP connections.

Behavior

On July 12, 2001, the Code-Red I 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 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.

On the 20th of every month, the worm is programmed to stop infecting other machines and proceed to its next attack phase in which it launches a Denial-of-Service attack against www.whitehouse.gov from the 20th to the 28th of each month. The worm is dormant on days of the month following the 28th.

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 I worm. The first version of the Code-Red worm (Code-Red I v1) caused little damage. Although the worm's attempts to spread itself consumed resources on infected machines and local area networks, it had little impact on global resources.

On August 4, 2001, an entirely new worm, Code Red II 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 I worm, the source code of the worm contained the string "Code Red II" which became the name of the new worm. Ryan Permeh and Marc Maiffret analyzed CodeRed II 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 I, Code Red II is not memory resident, so rebooting an infected machine does not eliminate Code Red II.

After rebooting the machine, the Code Red II worm begins to spread. If the host infected with Code Red II has Chinese (Taiwanese) or Chinese (PRC) as the system language, it uses 600 threads to probe other machines. On all other machines it uses 300 threads. Code Red II uses a more complex method of selecting hosts to probe than Code-Red I. Code Red II 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.

The Code Red II worm is much more dangerous than Code-Red I because Code Red II installs a mechanism for remote, administrator-level access to the infected machine. Unlike Code-Red I, Code Red II 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 (Denial-of-Service or otherwise).

• ***Detection Scheme***

In this section, we first characterize the spread of the Code Red I and Code Red II worms, then examine the properties of the infected host population, and finally determine the rate at which infected hosts are repaired.

To determine the rate of host infection, we recorded the time of the first attempt of each infected host to spread the worm.

• ***Deactivation Rate***

During the course of a particular day, a few initially infected machines were patched,

rebooted, or filtered and consequently ceased to probe networks for vulnerable hosts. We consider a host that was previously infected to be inactive. The majority of hosts stopped probing in the last hour before midnight UTC. At midnight, the worm was programmed to switch from an "infection phase" to an "attack phase", so the large rise in host inactivity is due to this design. The end of day phase change can be seen clearly, which shows the number of newly inactive hosts per minute. Because the Code Red II worm infects the same host population as Code-Red I v2, we neither expected nor measured an increase in the number of hosts probing our network once the Code Red II worm began to spread.

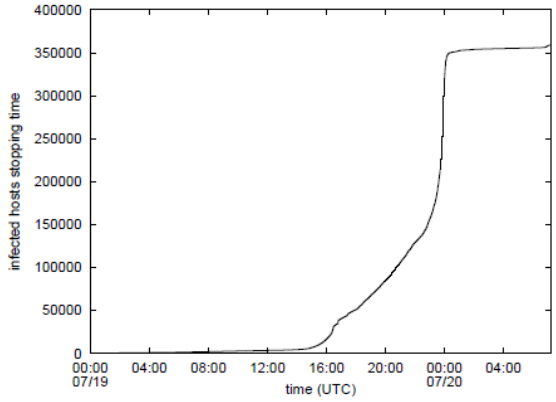

Fig 2: Cumulative total of deactivated Code-Red I infected hosts.

We also monitored no significant difference in the overall number of unsolicited TCP SYNs. No change in probe rate is apparent following the spread of Code Red II. Although Code Red II uses six times as many threads to spread as Code-Red I .Because our /8 network contained no susceptible hosts, the net probe rate we observe from Code Red II is the same as that of Code-Red I. Thus, we cannot distinguish hosts infected with Code Red II from those infected with Code-Red I. In their October 2001 study, Arbor Networks measured the ratio between Code-Red I and Code Red II probes to be 1:3.This 1:3 ratio may indicate the ratio between hosts infected with Code Red II versus Code Red I.

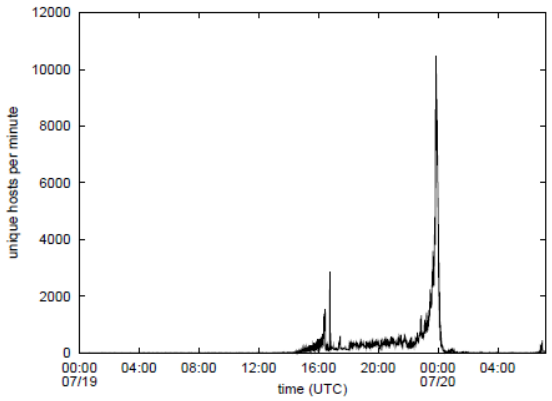

Fig 3: Rate of infected host deactivation in one minute periods.

3. Slammer Worm

Slammer, also known as Sapphire, was one of the smallest worms ever seen. It was found in January 2003 targeting Microsoft SQL Server 2000 or MSDE 2000. Slammer uses UDP port 1434 to exploit a buffer overflow in an MS SQL server. The code size is 376 bytes. Adding the UDP header makes the worm 404 bytes long in total. It uses a blind scan scheme where randomly generated numbers are used as IP addresses in searching for vulnerable hosts. To initialize the random number generator, Slammer uses the GetTickCount() function from Win32 API. Sometimes the random generator returns values that are broadcast addresses, such as a.b.c.255, and causes all the hosts in that network to receive the worm packets, making the spread of the Slammer worm more rapid. Like most UDP worms, Slammer is self-carried and has a monomorphic payload. Slammer does not write to the disks of infected machines, it only overloads the victim systems and slows down traffic.

• *Behaviour*

The worm's spreading strategy uses random scanning .It randomly selects IP addresses, eventually finding and infecting all susceptible hosts. Random-scanning worms initially spread exponentially, but their rapid new-host infection slows as the worms continually retry infected or immune addresses. Thus, as with the Code Red worm, Slammer's infected-host proportion follows a classic logistic form of initial exponential growth in a finite system.

• ***Detection Scheme***

While Slammer spread nearly two orders of magnitude faster than Code Red, it probably infected fewer machines. Both worms use the same basic scanning strategy to find vulnerable machines and transfer their exploitive payloads.

However, they differ in their scanning constraints. While Code Red is latency-limited, Slammer is bandwidth-limited, enabling Slammer to scan as fast as a compromised computer can transmit packets or a network can deliver them. For a random-scanning worm to be effective, it needs a good source of random numbers to select new attack targets. Slammer's random-number generator has some interesting deficiencies that make our analysis difficult and, perhaps, have implications for future worms. Slammer uses a linear congruent or power residue, pseudo random number generation (PRNG) algorithm.

These algorithms take the form: $x' = (x \times a + b) \bmod m$, where x' is the new pseudo random number to be generated, x is the last pseudo random number generated, m represents the range of the result, and a and b are carefully chosen constants. Linear congruent generators are very efficient and, with appropriate values of a and b have reasonably good distributional properties (although they are not random from a sequential standpoint). Slammer's author intended to use a linear congruent parameterization that Microsoft popularized, $x' = (x \times 214013 + 2531011) \bmod 232$. However, we found two implementation mistakes. First, the author substituted a different value for the 2531011 increment value: hex 0xFFD9613C. This value is equivalent to –2531012 when interpreted as a two's-complement decimal. So, it seems likely that the conversion to a negative number was an error (the author seems to have forgotten that creating a negative number in twos complement requires inverting and adding 1, not simply inverting), and probably that the author intended to use the SUB instruction to compensate for the resulting negative number, but mistakenly used ADD instead. The negative constant would be more desirable in the code, as this would eliminate any null (all zero) characters from the worm's code. The result is that the increment is always even. The author's second mistake was to misuse the OR instruction, instead of XOR, to clear an important register. This error left the register's previous contents intact.

These mistakes significantly reduce the generator's distribution quality. Because b is even and the salt is always 32-bit aligned, the least-significant two bits are always zero. Interpreted as a big-endian IP address (the most significant value in the sequence is stored at the lowest storage address), this ensures that the 25th and 26th bits in the scan address (the upper octet) remain constant in any worm execution instance. Similar weaknesses extend to the 24th bit of the address, depending on the un cleared register's value. Moreover, with the incorrectly chosen increment, any particular worm instance cycles through a list of addresses significantly smaller than the actual Internet address space.

Thus, many worm instances will never probe our monitored addresses because none of them are contained in the worm's scan cycle. Combined with the size of our monitored address space, 3 these mistakes prevent us from accurately measuring the number of infected hosts during the first minutes of the worm's spread.

$x' = (x \times 214013 + 2531011)$

Slammer will or will not include entire /16 address blocks (large contiguous address ranges) in a cycle, because the last two bits of the first address byte never change. We were able to assemble lists of the address blocks in each cycle for each value of the salt (cycle structure depends on salt value). Fortunately, the probability of choosing a particular cycle is directly proportional to the size of the cycle if the initial seed is selected uniformly at random.

If we looked at many randomly seeded worms, it is likely that all Internet addresses would be probed equally. Thus, we can accurately estimate the worm's scanning rate during the infection process by monitoring relatively small address ranges. We can estimate the percentage of the Internet that is infected because the probing will cover all Internet addresses.

If not for the higher-quality numbers in the initial seed, these flaws would prevent the worm from reaching large portions of the Internet

address space, no matter how many hosts were infected. For the same reason, these flaws also could bias our measurements because even though our data come from several different networks, there is a small chance that those particular networks were disproportionately more or less likely to be scanned.

3. Witty Worm

The Witty worm was released in March 2004, targeting buffer overflow vulnerability in several Internet Security Systems (ISSs), including Real Secure Server Sensor, Real Secure Desktop, and Black ICE. Witty took advantage of a vulnerability of the ISS Protocol Analysis Module (PAM) used for ICQ instant messaging. Witty is a self-carried monomorphic UDP worm that employs a blind target finding scheme. It sends out UDP packets to 20,000 random generated IP addresses on random destination ports from source port 4000, with a random packet size ranging between 768– 1307 bytes. The code size of Witty is only 637 bytes, and the rest of the payload is padded with data from system memory. This padding does not change the monomorphic format of Witty. The payload contains the text "(^.^) insert witty message here (^.^)," which is why it is named Witty. Witty randomly writes data onto the disk of infected machines. It is harder to detect Witty worms than worms with fixed size packets targeting fixed destination port numbers because of its random characteristics. The size of Witty worms is larger than Slammer worms, but they spread faster than Slammer. This proves that size is not always the bottleneck for the spreading of UDP worms. Another significance of the Witty worm is that it was the first known worms distributed using botnets.

Behavior

- It was the first widely propagated Internet worm to carry a destructive payload.
- It started in an organized manner with an order of magnitude more ground-zero hosts than any previous worm.
- It represents the shortest known interval between vulnerability disclosure and worm release. It began spreading the day after the ISS vulnerability was publicized.
- It spread through a host population in which every compromised host was proactive in securing its computers and networks.
- It spread through a population almost an order of magnitude smaller than that of previous worms, demonstrating worms' viability as an automated mechanism to rapidly compromise machines on the Internet, even in niches without a software monopoly.

Detection Scheme

In this section, we in detail explain the technical aspects of monitoring Witty worm.

Network telescope

Because Internet worm victims span diverse geographic and topological locations, the overall impact of a worm is difficult to measure from a single viewpoint. The University of California, San Diego (UCSD) Network Telescope consists of a large piece of globally announced IPv4 address space that we have instrumented to monitor network security events. The telescope contains almost no legitimate hosts, so inbound traffic to nonexistent machines is always anomalous in some way.

Because the network telescope contains approximately 1/256th of all IPv4 addresses, we receive roughly one out of every 256 packets sent by an Internet worm with an unbiased random number generator. Because we are uniquely situated to receive traffic from every worm-infected host, we provide a global view of the spread of many Internet worms.

ISS vulnerability

Several ISS firewall products contain a protocol analysis module (PAM) to monitor application traffic. The PAM routine in version 3.6.16 of isspam1.dll analyzed ICQ server traffic and assumes that incoming packets on port 4000 are ICQv5 server responses and that this code contains a series of buffer overflow vulnerabilities. eEye discovered this vulnerability on 8 March 2004 and announced it with ISS 10 days later. ISS released an alert, warning users of a possibly exploitable security hole and providing updated software versions that were not vulnerable to the buffer overflow attack.

Witty worm details

Once Witty infects a host, the host sends 20,000 packets by generating packets with a random destination IP address, a random size between 796 and 1,307 bytes, and a destination port. The packets' small size ensures fast transmission; as well, they are unlikely to be too big, for any network segment that they traverse , if they were too big, they would be broken into smaller pieces in the network, which could slow the spread of the worm. Because the Witty packets sizes are random, they are more difficult to filter than fixed sized packets and complicate simple blocking measures that limit or prevent the worm's spread. If they had been a fixed size, it would have been easier to quickly block Witty traffic to limit or prevent the worm's spread. The worm payload of 637 bytes is padded with data from system memory to fill the random size, and a packet is sent out from source port 4000. After Witty sends the 20,000 packets, it seeks out a random point on the hard disk and writes 65kbytes of data from the beginning of iss-pam1.dll to the disk. After closing the disk, Witty repeats this process until the machine is rebooted or it permanently crashes.

Table 1: Existing Internet Worm Implementations

Worm	Target Finding Scheme	Propagation Scheme	Transmission Scheme
Morris	Blind	Self-carried	TCP
Code-Red	Blind*	Self-carried	TCP
Slammer	Blind	Self-carried	UDP
Witty	Blind	Botnet	UDP
*Code-Red II focused on local subnet scan			

CONCLUSION

We studied a new class of worms, which has the capability of self-propagation and further avoid them by detection. Our investigation showed that worms successfully propagate in the time domain as well as frequency domain. Based on observation, we identified various detection schemes to detect them. Our evaluation data showed that our schemes achieved superior detection performance against the worms in comparison with existing representative detection schemes.

An ideal system should use a combination of schemes to have more comprehensive coverage. Different detection schemes are useful at different levels of implementation. This paper lays the foundation for ongoing studies of "smart" worms that intelligently adapt their propagation patterns to reduce the effectiveness of countermeasures.

REFERENCES

1. Pele Li, Mehdi Salour, And Xiao Su, San Jose State Uuniversity, "A Survey of Internet Worm Detection and Containment 1ST QUARTER 2008, VOLUME 10, NO. 1
2. Wei Yu, Xun Wang, Prasad Calyam, Dong Xuan, and Wei Zhao, Fellow, IEEE Transaction on Dependable and Secure Computing, Vol 8, No. 3, May/June 2011 – " Modeling and Detection of Camouflaging Worm.
3. D. Moore, C. Shannon, and J. Brown, "Code-Red: A Case Study on the Spread and Victims of an Internet Worm," Proc. Second Internet Measurement Workshop (IMW), Nov. 2002.
4. D. Moore, V. Paxson, and S. Savage, "Inside the Slammer Worm," Proc. IEEE Magazine of Security and Privacy, July 2003.
5. E. Spafford, "The Internet Worm Program: An Analysis," *Comp.Commun. Rev.*, 1989.
6. "Morris (Computer Worm)," retrieved July 2007, http://en.wikipedia.org/wiki/Morris_wormM. Young, The Technical Writer's Handbook. Mill Valley, CA: University Science, 1989.
7. G. P. Schaffer, "Worms and Viruses and Botnets, Oh My! Rational Responses to Emerging Internet Threats," *IEEE Sec. & Privacy*, vol. 4, 2006, pp. 52–58.