1

An Inside Look at Botnets

Paul Barford Vinod Yegneswaran
{pb,vinod}@cs.wisc.edu

Computer Sciences Department
University of Wisconsin, Madison

Abstract

The continued growth and diversification of the Internet has been accompanied by an increasing prevalence of attacks and intrusions [40]. It can be argued, however, that a significant change in motivation for malicious activity has taken place over the past several years: from vandalism and recognition in the hacker community, to attacks and intrusions for financial gain. This shift has been marked by a growing sophistication in the tools and methods used to conduct attacks, thereby escalating the network security arms race.

Our thesis is that the *reactive* methods for network security that are predominant today are ultimately insufficient and that more *proactive* methods are required. One such approach is to develop a foundational understanding of the mechanisms employed by malicious software (malware) which is often readily available in source form on the Internet. While it is well known that large IT security companies maintain detailed databases of this information, these are not openly available and we are not aware of any such open repository. In this paper we begin the process of codifying the capabilities of malware by dissecting four widely-used Internet Relay Chat (IRC) botnet codebases. Each codebase is classified along seven key dimensions including botnet control mechanisms, host control mechanisms, propagation mechanisms, exploits, delivery mechanisms, obfuscation and deception mechanisms. Our study reveals the complexity of botnet software, and we discusses implications for defense strategies based on our analysis.

1.1 Introduction

Software for malicious attacks and intrusions (malware) has evolved a great deal over the past several years. This evolution is driven primarily by the desire of the authors (black hats) to elude improvements in network defense systems and to expand and enhance malware capabilities. The evolution of malcode can be seen both in terms of variants of existing tools (*e.g.*, there are over 580 variants of the Agobot malware

since it's first release in 2002 [7]) and in the relatively frequent emergence of completely new codebases (*e.g.,* there were six major Internet worm families introduced in 2004: Netsky, Bagle, MyDoom, Sassser, Korgo and Witty as well as the Cabir virus - the first for cell phones [1]).

While worm outbreaks and DoS attacks have been widely reported in the popular press and evaluated extensively by the network and security research communities (*e.g.,* [16, 27–29]), perhaps the most serious threat to the Internet today are collections of compromised systems that can be controlled by a single person. These *botnets* have actually been in existence for quite some time and trace their roots to the Eggdrop bot created by Jeff Fisher for benign network management in 1993. High level overviews of malicious botnet history and their basic functionality can be found in [11, 31]. Over the years botnet capability has increased substantially to the point of blurring the lines between traditional categories of malware. There have been numerous reports of botnets of over one hundred thousand systems (although the average size appears to be dropping) and the total number of estimated systems used in botnets today is in the millions [17, 19, 23].

A plausible reason for the rise of malicious botnets is that the basic motivations for malicious activity are shifting. In the past, the primary motivations for attacks appear to have been simple (but potent) "script kiddie" vandalism and demonstrations of programming prowess in the black hat community. However, there are an increasing number of reports of for-profit malicious activity including identity theft and extortion that may be backed by organized crime (*e.g.,* [2, 35, 37]). This trend toward an economic motivation is likely to catalyze development of new capabilities in botnet code making the task of securing networks against this threat much more difficult.

The thesis for our work is that effective network security in the future will be based on detailed understanding of the mechanisms used by malware. While this high level statement does not represent a significant departure from what has been the modus operandi of the IT security industry for some time, unfortunately, data sharing between industry and research to date has not been common. We argue that greater openness and more detailed evaluations of the mechanisms of malware are required across the network security research community. In some respects this broadens the Internet Center for Disease Control vision outlined by Staniford *et al.* in [34]. We advocate analysis that includes both static inspection of malware source code when it is available and dynamic profiling of malware executables in a controlled environment. An argument for the basic feasibility of this approach is that a good deal of malware is, in fact, available on line (*e.g.,* [26]) and there are emerging laboratory environments such as WAIL [10] and DETER [15] that enable safe evaluation of executables. It is important to emphasize that these analyses are meant to *complement* the ongoing empirical measurement-based studies (*e.g.,* [9, 30, 36]) which provide important insight on how malware behaves in the wild, and are critical in identifying new instances of outbreaks and attacks.

This paper presents a first step in the process of codification of malware mechanisms. In particular, we present an initial breakdown of four of the major botnet source codebases including Agobot, SDBot, SpyBot and GT Bot. We conduct this

analysis by creating a taxonomy of seven key mechanisms and then describe the associated capabilities for specific instances of each bot family. Our taxonomy emphasizes botnet architecture, control mechanisms, and methods for propagation and attack. Our objectives are to highlight the richness and diversity of each codebase, to identify commonalities between codebases and to consider how knowledge of these mechanisms can lead to development of more effective defense mechanisms.

A summary of our findings and their implications are as follows:

- **Finding:** The overall architecture and implementation of botnets is complex, and is evolving toward the use of common software engineering techniques such as modularity. **Implication:** The regularization of botnet architecture provides insight on potential extensibility and could help to facilitate systematic evaluation of botnet code in the future.
- **Finding:** The predominant remote control mechanism for botnets remains Internet Relay Chat (IRC) and in general includes a rich set of commands enabling a wide range of use. **Implication:** Monitors of botnet activity on IRC channels and disruption of specific channels on IRC servers should continue to be an effective defensive strategy for the time being.
- **Finding:** The host control mechanisms used for harvesting sensitive information from host systems are ingenious and enable data from passwords to mailing lists to credit card numbers to be gathered. **Implication:** This is one of the most serious results of our study and suggests design objectives for future operating systems and applications that deal with sensitive data.
- **Finding:** There is a wide diversity of exploits for infecting target systems written into botnet codebases including many of those used by worms that target well known Microsoft vulnerabilities. **Implication:** This is yet additional evidence that keeping OS patches up to date is essential and also informs requirements for network intrusion detection and prevention systems.
- **Finding:** All botnets include denial of service (DoS) attack capability. **Implication:** The specific DoS mechanisms in botnets can inform designs for future DoS defense architectures.
- **Finding:** Shell encoding and packing mechanisms that can enable attacks to circumvent defensive systems are common. However, Agobot is the only botnet codebase that includes support for (limited) polymorphism. **Implication:** A significant focus on methods for detecting polymorphic attacks may not be warranted at this time but encodings will continue to present a challenge for defensive systems.
- **Finding:** All botnets include a variety of sophisticated mechanisms for avoiding detection (*e.g.*, by anti-virus software) once installed on a host system. **Implication:** Development of methods for detecting and disinfecting compromised systems will need to keep pace.
- **Finding:** There are at present only a limited set of propagation mechanisms available in botnets with Agobot showing the widest variety. Simple horizontal and vertical scanning are the most common mechanism. **Implication:** The specific

propagation methods used in these botnets can form the basis for modeling and simulating botnet propagation in research studies.

The remainder of this paper is structured as follows. While there have been relatively few studies of botnets in the research literature to date, we discuss other related work in Section 1.2. In Section 1.3 we present our taxonomy of botnet code and the results of evaluating four instances of botnet source code. In Section 1.4 we summarize our work and comment on our next steps.

1.2 Related Work

Empirical studies have been one of the most important sources of information on malicious activity for some time. Moore *et al.* characterized the Code Red I/II worm outbreaks in [29] and the Sapphire/Slammer worm outbreak [27] providing key details on propagation methods and infection rates. Recently, Kumar *et al.* show how a broad range of details of the Witty worm outbreak can be inferred using information about that malware's random number generator [24]. In [40], firewall and intrusion detection system logs collected from sites distributed throughout the Internet are used to characterize global attack activity. Several recent studies have demonstrated the utility of unused address space monitors (honeynets) [21] that include active response capability as a means for gathering details on network attacks [9, 30, 39]. Honeynet measurement studies have also provided valuable information on botnet activity [18, 39]. Cooke *et al.* discuss the potential of correlating data from multiple sources as a means for detecting the botnet command and control traffic in [12]. Finally, the virtual honeyfarm capabilities described in [38] could prove to be very useful for botnet tracking in the future.

As we advocated in the prior section, another way to study malware is to gather and then decompose instances of both source code (many instances of malware source code can be found by searching the Web and Usenet news groups) and executable code (executables can be gathered by enhancing honeynet environments). There are standard tools available for reverse engineering executables including disassemblers, debuggers and system monitors such as [4–6]. Despite the capabilities of these tools, the complexity and deception techniques of certain instances of malware executables often complicate this analysis [3]. Likewise, there are many tools available for static analysis of source code such as [13, 14]. While these tools are often focused on the problems of identifying run time errors and security vulnerabilities, the general information they provide such as parse trees, symbol tables and call graphs could be valuable in our malware analysis. While we present a simple taxonomy of malware mechanisms in this paper, we look forward to using both static and dynamic analysis tools for in depth study in the future.

1.3 Evaluation

Our process of codification of malware begins with a comparison of four botnet families: Agobot, SDBot, SpyBot and GT Bot. These were selected based on the age of their first known instances, the diversity in their design and capabilities, and

reports in the popular press, commercial and research communities identifying these as the most commonly used bot families. While each of these families have many versions and variants, for this study we evaluate one version of source code from each: Agotbot (4.0 pre-release), SDBot (05b) and SpyBot (1.4). GT Bot variants are commonly listed with extensions after the word "Bot" *e.g.,* "GT Bot Foo" – we evaluated the "GT Bot with DCOM" version of this code.

The attributes we consider in our analysis include: (i) architecture, (ii) botnet control mechanisms, (iii) host control mechanisms, (iv) propagation mechanisms, (v) target exploits and attack mechanisms, (vi) malware delivery mechanisms, (v) obfuscation methods, and (vii) deception strategies. This taxonomy was developed based on our goal of improving both host and network-based defensive systems by exploiting knowledge of basic features of botnet systems.

1.3.1 Architecture

Architecture refers to the design and implementation characteristics of bot code. Architecture is readily analyzed from source code and includes assessment of the overall organization, data design, interface design and component design of the system. An important additional objective in this analysis is to assess the potential long term viability of each bot family by considering how each codebase might be extended to include new functionality.

- **Agobot:** The earliest references to Agobot that we could find were in the October, 2002 time frame [32]. There are now many hundreds of variants of this code which is also commonly referred to as Phatbot. It is arguably the most sophisticated and best-written source code among the four families we evaluated. A typical source bundle is around 20,000 lines of C/C++. The bot consists of several high level components including, (i) an IRC-based command and control mechanism, (ii) a large collection of target exploits, (iii) the ability to launch different kinds of DoS attacks, (iv) modules that support shell encodings and limited polymorphic obfuscations, (v) the ability to harvest the local host for Paypal passwords, AOL keys and other sensitive information either through traffic sniffing, key logging or searching registry entries, (vi) mechanisms to defend and fortify compromised systems either through closing back doors, patching vulnerabilities or disabling access to anti-virus sites, and (vii) mechanisms to frustrate disassembly by well known tools such as SoftIce, Ollydbg and others. Agobot has a monolithic architecture, demonstrates creativity in design, and adheres to structured design and software engineering principles through its modularity, standard data structures and code documentation.
- **SDBot:** The earliest references to SDBot that we could find were in the October, 2002 time frame [33]. There are now hundreds of variants of this code that provide a wide range of capabilities. In contrast with Agobot, SDBot is a fairly simple, more compact instance of bot code written in slightly over 2,000 lines of C. The main source tree does not include any overtly malicious code modules such as target exploits or DoS capabilities, and is published under GPL. SDBot

primarily provides a utilitarian IRC-based command and control system. However, the code is obviously easy to extend, and a large number of patches are readily available that provide more sophisticated malicious capabilities such as scanning, DoS attacks, sniffers, information harvesting routines and encryption routines. This organization facilitates generation of custom botnets with specialized capabilities that suit a specific botmaster. We speculate that an important motivation for this patch-style dissemination strategy is diffusion of accountability. We easily found around 80 patches for SDBot [1] on the Web, not all of which were malicious.

- **SpyBot:** The earliest references to SpyBot that we could find were in the April, 2003 time frame [25]. Like Agobot and SDBot there are now hundreds of variants of SpyBot. The codebase is relatively compact, written in under 3,000 lines of C. Much of SpyBot's command and control engine appears to be shared with SDBot, and it is likely, in fact, that it evolved from SDBot. However, unlike SDBot, there is no explicit attempt to diffuse accountability or to hide the malicious intent of this codebase. The version of SpyBot that we evaluated includes NetBIOS/Kuang/Netdevil/KaZaa exploits, scanning capability, and modules for launching flooding attacks. Overall, the codebase for Spybot is efficient, but does not exhibit the modularity or breadth of capabilities of Agobot.
- **GT Bot:** The earliest references to GT Bot that we could find were in the April, 1998 time frame [8]. At present there are well over a hundred variants of GT (Global Threat) Bot which is also referred to as Aristotles. GT Bot's design is quite simple, providing a limited set of functions based on the scripting capabilities of mIRC which is a widely used shareware IRC client for Windows. mIRC provides functionality for writing event handlers that responds to commands received by remote nodes. GT Bot also includes the HideWindow program which keeps the bot hidden on the local system. While this bot has proved easy to modify, there is nothing that suggests it was designed with extensibility in mind. GT Bot capabilities including port scanning, DoS attacks, and exploits for RPC and NetBIOS services. GT Bot scripts are commonly stored in a file called *mirc.ini* on compromised local hosts. However GT Bot is often packaged with its own version of the *mIRC.exe* that has been hex-edited to include other configuration files. Other useful pieces of software that are often packaged with GT Bot include BNC (pronounced "bounce") which is a proxy system that allows users to bounce through shells to a IRC server providing anonymity and DoS protection, and *psexec.exe* (SysInternals) which is a utility that facilitates remote process execution. Based on the limited capabilities in GT Bot, it appears that different versions have been generated for specific malicious intent, instead of general enhancement of the code to provide a broad set of capabilities. As the name suggests, the "with DCOM" version of GT Bot that we evaluated includes DCOM exploit capabilities.

[1] These are not UNIX-style patches, rather they are simply well-commented source code fragments that can be copied and inserted before recompilation.

Implications: While bot codebases vary in size, structure, complexity, and implementation approach, there appears to be a convergence in the set of functions that are available (this will be further highlighted in subsequent sections of this report). This suggests the possibility that defensive systems may be eventually be effective across bot families. Further, as demonstrated by the fact that there are so many variants in each codebase, all of the bot families are at least somewhat extensible. However, we project that over the next several years, due to economic motivations, capabilities and open availability, the Agobot codebase is likely to become dominant. It's modular design makes it easy to extend, and we anticipate future enhancements such as improved command and control systems (*e.g.*, peer-to-peer) and additional target exploits. While an open-source-like approach to Agobot's development is somewhat daunting, it's open availability means that it can be examined for elements which can be exploited by defensive systems.

1.3.2 Botnet Control Mechanisms

Botnet control refers to the command language and control protocols used to operate botnets remotely after target systems have been compromised. The command and control mechanisms for the bots that we evaluated are all based on IRC. Thus, an understanding of that system (*e.g.*, see IETF RFC #1459 which defines IRC) will help to make sense out of the botnet commands detailed in this section. In general, there is a broad range of commands that are available. These include directing botnets to deny service, send spam, phish, forward sensitive information about hosts, and look for new systems to add to the botnet.

The most important reason for understanding the details of the communication mechanisms is that their disruption can render a botnet useless. For example, by sniffing for specific commands in IRC traffic, network operators can identify compromised systems, and IRC server operators can shutdown channels that are used by botnets (this is commonly done today). Additionally, knowledge of these mechanisms can be used in development of large botnet monitors (*e.g.*, via active honeynet systems), and it also facilitates the process of detecting new variants. While control mechanisms occasionally change between versions, there is strong commonality within each family we analyzed. This bodes well for continued focus on these mechanisms when designing network defenses against botnets.

- **Agobot:** The command and control system implemented in Agobot is a derivative of IRC. The protocol used by compromised systems to establish connections to control channels is standard IRC. The command language consists of both standard IRC commands and specific commands developed for this bot. Details of the command language are summarized in Table 1.1. The bot command set includes directives that request the bot to perform a specific function *e.g.*, **`bot.open`** which opens a specific file on the host. The control variables are used in conjunction with the **`cvar.set`** command to turn on/off features or otherwise manipulate fields that affect modes of operation *e.g.* **`ddos_max_threads`** which directs the bot to SYN flood a specified host using a maximum number of threads.

Table 1.1. Partial listing of the Agobot command and control language. The "variables" are passed as parameters to the **`cvar.set`** set command.

Variable	**Description**
bot_ftrans_port	Set bot - file transfer port
bot_ftrans_port_ftp	Set bot - file transfer port for FTP
si_chanpass	IRC server information - channel password
si_mainchan	IRC server information - main channel
si_nickprefix	IRC server information - nickname prefix
si_port	IRC server information - server port
si_server	IRC server information - server address
si_servpass	IRC server information - server password
si_usessl	IRC server information - use SSL ?
si_nick	IRC server information - nickname
bot_version	Bot - version
bot_filename	Bot - runtime filename
bot_id	Bot - current ID
bot_prefix	Bot - command prefix
bot_timeo	Bot - timeout for receiving (in milliseconds)
bot_seclogin	Bot - enable login only by channel messages
bot_compnick	Bot - use the computer name as a nickname
bot_randnick	Bot - random nicknames of letters and numbers
bot_meltserver	Bot - melt the original server file
bot_topiccmd	Bot - execute topic commands
do_speedtest	Bot - do speed test on startup
do_avkill	Bot - enable anti-virus kill
do_stealth	Bot - enable stealth operation
as_valname	Autostart - value name
as_enabled	Autostart - enabled
as_service	Autostart - start as service
as_service_name	Autostart - short service name
scan_maxthreads	Scanner - maximum number of threads
scan_maxsockets	Scanner - Maximum number of sockets
ddos_maxthreads	DDoS - maximum number of threads
redir_maxthreads	Redirect - maximum number of threads
identd_enabled	IdentD - enable the server
cdkey_windows	Return windows product keys on cdkey.get
scaninfo_chan	Scanner - output channel
scaninfo_level	Info level 1 (less) - (3) more
spam_aol_channel	AOL spam - channel name
spam_aol_enabled	AOL spam - enabled ?
sniffer_enabled	Sniffer - enabled ?
sniffer_channel	Sniffer - output channel
vuln_channel	Vulnerability daemon sniffer channel
inst_polymorph	Installer - polymorphoic on install ?
Command	**Description**
bot.about	Displays information (*e.g.*, version) about the bot code
bot.die	Terminates the bot
bot.dns	Resolves IP/hostname via DNS
bot.execute	Makes the bot execute a specific .exe
bot.id	Displays the ID of the current bot code
bot.nick	Changes the nickname of the bot
bot.open	Opens a specified file
bot.remove	Removes the bot from the host
bot.removeallbut	Removes the bot if ID does not match
bot.rndnick	Makes the bot generate a new random nickname
bot.status	Echo bot status information
bot.sysinfo	Echo the bot's system information
bot.longuptime	If uptime $>$ 7 days then bot will respond
bot.highspeed	If speed $>$ 5000 then bot will respond
bot.quit	Quits the bot
bot.flushdns	Flushes the bot's DNS cache
bot.secure	Delete specified shares and disable DCOM
bot.unsecure	Enable specified shares and enables DCOM
bot.command	Executes a specified command with system()

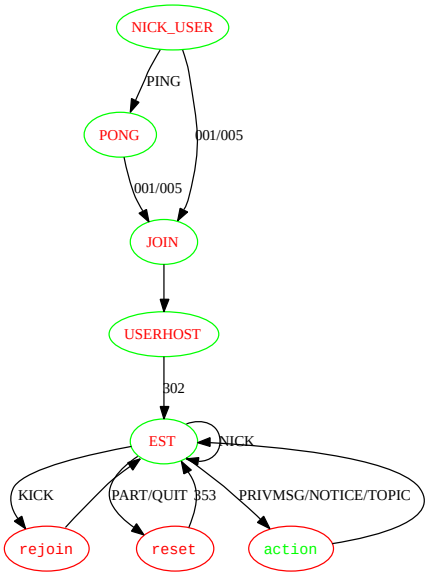

Fig. 1.1. Typical interaction between an SDBot and IRC server.

- **SDBot:** The command language implemented in SDBot is essentially a lightweight version of IRC. Figure 1.1 illustrates the state transition sequence of a compromised host interacting with an IRC server. The bot begins by establishing a connection to the IRC server through the following steps: (i) send NICK (name) and USER (name) to login to the server, (ii) if a PING is received, respond with a PONG, (iii) when connected to the server (*i.e.*, return code 001 or 005), send a JOIN message followed by a USERHOST request to obtain the hostname, (iv) wait for a 302 response that indicates a connection is established, (v) listen and react to commands sent by the master which can include the following:
  1. KICK: the bot rejoins the channel if it is kicked off. Otherwise the bot resets the master if the master is kicked.
  2. NICK: if master's nickname is replaced, then it is updated on the bot.
  3. PART (or QUIT): resets the master if the master parts or quits.
  4. 353: return code that indicates that the bot has successfully joined the IRC channel.

  The bot then expects all other commands will be sent as part of the PRIVMSG, NOTICE or TOPIC IRC messages. The commands available in SDBot are listed in Table 1.2. Additional features supported by SDBot but absent from Agobot include IRC cloning and spying. Cloning is when a bot connects to an IRC channel multiple times. This can be used to deny service on a particular IRC server. Spying is simply the act of logging activity on a specified IRC channel.

Table 1.2. Partial listing of the SDBot command language. These commands are passed to bots via the PRIVMSG, NOTICE or TOPIC IRC commands.

Command	**Description**
about	Displays information about the bot code
action <channel/user>, <text>	Perform specified action on the channel
addalias <alias, command>	Add an alias
aliases	Return a current list of aliases
cycle<N> <channel>	Leave channel and return after N seconds
die	Kill all threads, close IRC connection and stop running
disconnect	Disconnect from channel and reconnect in 30 minutes
id	Return the bot ID
join <channel> <key>	Join specified channel with specified key
log	Return a log of connections, logins and time stamps
nick <newnick>	Changes bot's nickname
part	Part the specified channel
prefix	Temporary change to bot's IP prefix
quit	Quit the channel, kill threads and close the bot
raw <text>	Send the following text to server
reconnect	Disconnect and reconnect to receive new nickname and ID
repeat <numtimes> <command>	Act as if command was received numtimes
rndnick	Change to random nickname
server <servername>	Temporarily changes bot's IRC server
status	Echo with version number and bot's uptime
Clones and Spies	
clone <server><port><channel>	Create clone on specified channel
c_rndnick <threadnum>	Causes clone to change to random nickname
c_raw <threadnum> <text>	Causes clone to send text to server
c_quit <threadnum>	Causes the clone/spy to quit the IRC server
c_nick <threadnum> <nick>	Causes the clone/spy to change its nickname
c_privmsg <threadnum> <user> <text>	Causes clone/spy to send message channel with text
c_part <threadnum> <channel>	Causes clone/spy to part channel
c_mode <threadnum> <channel> <mode> <user>	Causes clone to set a channel or user mode
c_join <threadnum> <channel>	Causes clone/spy to join channel
c_action <threadnum> <channel> <text>	Causes clone/spy to perform an action to the specified channel.
spy <nick> <server> <port> <channel>	Creates spy with specified nickname on server,port,channel

- **SpyBot:** The command language implemented in SpyBot is quite simple and essentially represents a subset of the SDBot command language. The commands available in SpyBot are listed in Table 1.3. The IRC connection set up protocol for SpyBot is the same as SDBot, and the mechanisms to pass and execute commands on bots are also identical.
- **GT Bot:** Like the other families, GT Bot uses IRC as its control infrastructure. The command language implemented in GT Bot is the simplest of all of those that we evaluated, but it varies quite a bit across versions within this family. This is likely due to the architecture of GT Bot which facilitates creation of versions with specific intent instead of developing a broad range of capabilities within a single line of the codebase. We provide a list of the commands supported by the GTBot-with-dcom source code used in our analysis in Table 1.4.

Implications: Understanding command and control systems has direct and immediate implications for creation of methods and systems to disrupt botnets. The continued reliance on IRC as the foundation for botnet command and control means that IRC server operators can play a central role in blocking botnet traffic (anecdotally, they already do). However, monitoring and shutting down botnet channels by hand is arduous, and automated mechanisms for identifying botnet traffic are re-

Table 1.3. Partial listing of the SpyBot command language. These commands are passed to bots via the PRIVMSG, NOTICE or TOPIC IRC commands.

Command	Description
login < password >	Login to the bot
info	Provides information about host system
passwords	Lists the RAS passwords in MS Windows 9x versions
disconnect < secs >	Disconnect bot for t seconds (default is 30 minutes)
reconnect	Disconnect and then reconnect
server < new server addr >	Temporarily changes the bot's IRC server
quit	Quit the channel, kill threads and close bot
uninstall	Uninstalls the bot
redirect <in port> <host> <out port>	Redirect traffic from host to output port
raw <command>	Echo command to server
download <url> <filename>	Copy contents of url to filename
list <path+filter>	List c:\ * .*
spy	Redirects all traffic from the IRC server to the DCC chat
stopspy	Stops the spy
redirectspy	Redirects all traffic from the port redirect to the DCC chat
stopredirectspy	Stops redirect spy
loadclones <server> <port> <numclones>	Load numclones clones on server
killclones	Kills all the clones
rawclones <command>	Execute raw command on all clones

Table 1.4. Partial listing of the GT Bot command language. These commands are passed to bots via the PRIVMSG, NOTICE or TOPIC IRC commands.

Command	Description
!ver	Returns the version of the botnet
!info	Returns local host information e.g., OS, uptime, etc.
!scan <ip.*><port>	Scan specified address prefix on specified port
!portscan <IP><sport><eport>	Scan specified address across specified ports
!stopscan	Stops all scans
!packet <IP><number>	Start denial of service attack (ping.exe) of IP
!bnc	Execute commands specific to the bounce proxy system
!clone.*	Directs all IRC clone behavior (attacks, etc.)
!update<url>	Update version of bot code from a specified Web page
!-	Executes command on local host

quired. The botnet command languages outlined in this section can be used in the development of such systems and we project this will be a fruitful short term focus area. However, we anticipate that future botnet development will include the use of encrypted communication, eventually a movement away from IRC and adopt peer-to-peer style communication (some versions of Phatbot are already reported to have rudimentary P2P capability). While this will certainly make defending against botnets more difficult, botnet traffic may still be able to be identified via statistical finger printing methods.

1.3.3 Host Control Mechanisms

Host control refers to the mechanisms used by the bot to manipulate a victim host once it has been compromised. The general intent of host control is to fortify the local system against other malicious attacks, to disable anti-virus software, and to harvest sensitive information.

- **Agobot:** The set of host control capabilities provided in Agobot is quite comprehensive. These include, (i) commands to secure the system *e.g.*, close NetBIOS shares, RPC-DCOM, etc. (ii) a broad set of commands to harvest sensitive information (iii) `pctrl` commands to list the processes running on the host and kill specific processes (iv) `inst` commands to add or delete autostart entries. A summary of Agobot host control commands is provided in Table 1.5.

Table 1.5. Agobot host control commands.

Command	Description
harvest.cdkeys	Return a list of CD keys
harvest.emails	Return a list of emails
harvest.emailshttp	Return a list of emails via HTTP
harvest.aol	Return a list of AOL specific information
harvest.registry	Return registry information for specific registry path
harvest.windowskeys	Return Windows registry information
pctrl.list	Return list of all processes
pctrl.kill	Kill specified process set from service file
pctrl.listsvc	Return list of all services that are running
pctrl.killsvc	Delete/stop a specified service
pctrl.killpid	Kill specified process
inst.asadd	Add an autostart entry
inst.asdel	Delete an autostart entry
inst.svcadd	Adds a service to SCM
inst.svcdel	Delete a service from SCM

- **SDBot:** The host control capabilities provided in the base distribution of SDBot are somewhat limited. They include some basic remote execution commands and some capability to gather local information. The lack of host control capabilities in the basic distribution is likely due to SDBot's benign intent as described above. However, these capabilities can be easily enhanced through auxiliary patches and a large number of these are readily available. A summary of SDBot host control commands is provided in Table 1.6.

Table 1.6. SDBot host control commands.

Command	Description
download <url> <dest> <action>	Downloaded specified file and execute if action is 1
killthread <thread#>	Kill specified thread
update <url> <id>	If bot ID is different than current, download "sdbot executable" and update
sysinfo	List host system information (CPU/RAM/OS and uptime)
execute <visibility> <file> parameters	Run a specified program (visibility is 0/1)
cdkey/getcdkey	Return keys of popular games *e.g.*, Halflife, Soldier of Fortune etc.

- **SpyBot:** The host control capabilities included in SpyBot are relatively rich, and similar in most respects to what is provided by Agobot. These include commands for local file manipulation, key logging, process/system manipulation and remote command execution. A summary of the SpyBot host control commands is provided in Table 1.7.
- **GT Bot:** The set of host control commands provided in GT Bot is the most limited of all of the families we evaluated. The base capabilities include only gathering local system information and the ability to run or delete local files.

Table 1.7. SpyBot host control commands.

Command	Description
delete <filename>	Delete a specified file
execute <filename>	Execute a specified file
rename <origfilename> <newfile>	Rename a specified file
makedir <dirname>	Create a specified directory
startkeylogger	Starts the on-line keylogger
stopkeylogger	Stops the keylogger
sendkeys <keys>	Simulates key presses
keyboardlights	Flashes remote keyboard lights 50x
passwords	Lists the RAS passwords in Windows 9x systems
listprocesses	Return a list of all running processes
killprocess <processname>	Kills the specified process
threads	Returns a list of all running threads
killthread < number >	Kills a specified thread
disconnect <number>	Disconnect the bot for number seconds
reboot	Reboot the system
cd-rom <0/1>	Open/close cd-rom. cd-rom 1 = open, cd-rom 0 = close
opencmd	Starts cmd.exe (hidden)
cmd <command>	Sends a command to cmd.exe
get <filename>	Triggers DCC send on bot
update <url>	Updates local copy of the bot code

However, like SDBot, there are many versions of GT Bot that include diverse capabilities for malicious host control.

Implications: The capabilities and diversity of the host control mechanisms in botnets are frightening and have serious implications. First they underscore the need to patch and protect systems from known vulnerabilities. Second, they informs software development and the need for stronger protection boundaries across applications in operating systems. Third, the capabilities of gathering sensitive information such as Paypal passwords and software keys provide clear economic incentives for people to operate botnets and for sponsorship by organized crime.

1.3.4 Propagation Mechanisms

Propagation refers to the mechanisms used by bots to search for new host systems. Traditional propagation mechanisms consist of simple horizontal scans on a single port across a specified address range, or vertical scans on a single IP address across a specified range of ports. However, as botnet capability expands, it is likely that they will adopt more sophisticated propagation methods such as those proposed in [34].

- **Agobot:** The scanning mechanisms included in Agobot are relatively simple and do not extend very far beyond horizontal and vertical scanning. Agobot scanning is based on the notion of network ranges (network prefixes) that are configured on individual bots. When so directed, a bot can scan across a range or randomly select IP addresses within a range. However, the current scanning command set provides no means for efficient distribution of a target address space among a collection of bots. Table 1.8 provides a summary of the scanning commands in Agobot.

Table 1.8. Agobot propagation and scanning commands.

Command	Description
scan.addnetrange <IP range><priority>	Adds a network range to a bot
scan.delnetrange<IP range>	Deletes a network range from a bot
scan.listnetranges	Returns all network ranges registered with a bot
scan.clearnetranges	Clears all network ranges registered with a bot
scan.resetnetranges	Resets the network ranges to the localhost
scan.enable <module name>	Enables a scanner module *e.g.,* DCOM
scan.disable <module name>	Disables a scanner module
scan.startall	Directs all bots to start scanning their network ranges
scan.stopall	Directs all bots to stop scanning
scan.start	Directs all enabled bots start scanning
scan.stop	Directs all bots to stop scanning
scan.stats	Returns results of scans

- **SDBot:** As discussed in Section 1.3.1, by virtue of its benign intent, SDBot does not have scanning or propagation capability in its base distribution. However, many variants of SDBot include scanning and propagation capability. Among these, the scanning control interface is often quite similar to Agobot providing horizontal and vertical search capabilities. There are also instance where slightly more complex scanning methods are available. For example, the interface for a NetBIOS scanner for SDBot accepts starting and ending IP addresses as parameters and then randomly selects addresses between these two markers.
- **SpyBot:** The command interface for Spybot scanning is quite simple, consisting of horizontal and vertical capability. A typical example is given below:

```
Command:  scan <start IP address> <port> <delay>
                              <spreaders> <logfilename>
Example: scan 127.0.0.1 17300 1 netbios portscan.txt
```

  Scanning begins at the start address and opens MAX_PORTSCAN_SOCKETS_TO_USE sockets. The default value for this parameter is set to 20. Scanning then proceeds sequentially. The only spreader supported by the version of SpyBot that we evaluated is via NetBIOS.
- **GTBot:** As shown in Table 1.4, GT Bot only includes support for simple horizontal and vertical scanning.

Implications: There are several implications for bot propagation mechanisms. First, at present, botnets use relatively simple scanning techniques. This means that it may be possible to develop statistical finger printing methods to identify scans from botnets in distributed monitors. Second, scanning methods inform requirements for building and configuring network defenses based on firewalls and intrusion detection systems that consider scanning frequency. Finally, source code examination reveals detail of scanning mechanisms that can enable development of accurate botnet propagation models for analytic and simulation-based evaluation. We project that future versions of bot codebases will focus on propagation as an area of improvement, including both flash mechanisms and more stealthy mechanisms.

1.3.5 Exploits and Attack Mechanisms

Exploits refer to the specific methods for attacking known vulnerabilities on target systems. Exploits are usually attempted in conjunction with scanning for target hosts. In this section we discuss the specific exploit modules included in each bot, and other capabilities for launching remote attacks against target systems.

- **Agobot:** The most elaborate set of exploit modules among the families that we analyzed is included with Agobot. In contrast with the other bot families, Agobot's evolution has included an ever broadening set of exploits instead of individual versions with their own exploits. This increases Agobot's potential for compromising targeted hosts. The exploits in the version of Agobot that we evaluated include:
  1. Bagle scanner: scans for back doors left by Bagle variants on port 2745.
  2. Dcom scanners (1/2): scans for the well known DCE-RPC buffer overflow.
  3. MyDoom scanner: scans for back doors left by variants of the MyDoom worm on port 3127.
  4. Dameware scanner: scans for vulnerable versions of the Dameware network administration tool.
  5. NetBIOS scanner: brute force password scanning for open NetBIOS shares.
  6. Radmin scanner: scans for the Radmin buffer overflow.
  7. MS-SQL scanner: brute force password scanning for open SQL servers.
  8. Generic DDoS module: enables seven types of denial service attack against a targeted host. A list of the commands used to control these attacks is given in Table 1.9.

Table 1.9. Agobot DDos attack commands.

Command	Description
ddos.udpflood<target> <port><0=rand> <time>(secs) <delay>(ms)	Starts a UDP flood
ddos.synflood<host> <time> <delay> <port>	Starts a SYN flood
ddos.httpflood <url> <number> <referrer> <delay> <recursive>	Starts an HTTP flood
ddos.phatsyn <host> <time> <delay> <port>	Starts a PHAT SYN flood
ddos.phaticmp <host> <time> <delay>	Starts PHAT ICMP flood
ddos.phatwonk <host> <time> <delay>	Starts PHATwonk flood
ddos.targa3 <target> <time>(secs)	Start a targa3 flood
ddos.stop	stops all floods

- **SDBot:** As discussed in Section 1.3.1, by virtue of its benign intent, SDBot does not have any exploits packaged in its standard distribution. There are, however, numerous variants that include specific exploits. SDBot does include modules for sending both UDP and ICMP packets. While not overtly malicious, these can certainly be used for simple flooding attacks. Commands to control these capabilities are listed in Table 1.10. As might be expected, there are also numerous variants of SDBot that include different kinds of DDoS attack modules.
- **Spybot:** The exploits included in the version of Spybot that we evaluated only included attacks on NetBIOS open shares. However, as with SDBot, there are many variants that include a wide range of exploits. SpyBot's DDoS interface is

Table 1.10. SDBot commands which could be used for DDoS attacks.

Command	Description
udp <host> <# pkts> <pkt sz> <delay> <port>	Send a specified number of UDP packets
ping <host> <# pkts> <pkt sz> <timeout>	Send a specified number of ICMP echo packets

also closely related to SDBot and includes the capabilities for launching simple UDP, ICMP and TCP SYN floods.

- **GTBot:** As mentioned earlier, the exploit set for the GT Bot code that we evaluated was developed to include RPC-DCOM exploits. Like SDBot and Spybot, there are many variants of GT Bot that include other well known exploits. Our version of GT Bot only included capability to launch simple ICMP floods. However, there are many variants of GT Bot that have other DDoS capabilities such as UDP and TCP SYN floods.

Implications: The set of exploits packaged with botnets suggest basic requirements for host-based anti-virus systems and network intrusion detection and prevention signature sets. It seems clear that in the future, more bots will include the ability to launch multiple exploits as in Agobot since this increases the opportunity for success. The DDoS tools included in bots, while fairly straightforward, highlight the potential danger of large botnets. They also inform possibilities for DDoS protection strategies such as [22].

1.3.6 Malware Delivery Mechanisms

Packers and shell encoders have long been used in legitimate software distribution to compress and obfuscate code. The same techniques have been adopted in botnet malware for the same reasons. GT/SD/Spy Bots all deliver their exploit and encoded malware packaged in a single script. However, Agobot has adopted a new strategy for malware delivery based on separating exploits and delivery. The idea is to first exploit a vulnerability (*e.g.*, via buffer overflow) and open a shell on the remote host. The encoded malware binary is then uploaded using either HTTP or FTP. This separation enables an encoder to be used across exploits thereby streamlining the codebase and potentially diversifying the resulting bit streams.

In Figure 1.2 we provide an example of the shell-encoder used in Agobot for malware delivery. An important function of a shell-encoder is to remove null bytes (that terminate c-strings) from x86 instruction sequences. As can be seen in the Figure, the code begins with an XOR key value of 0x98 then checks to see if this results in a string without null characters. If the check fails, it simply tries successive values for the XOR key until it finds a value that works. This value is then copied over to the shell code at position ENCODER_OFFSET_XORKEY.

Implications: The malware delivery mechanisms used by botnets have implications for network intrusion detection and prevention signatures. In particular, NIDS/NIPS benefit from knowledge of commonly used shell codes and ability to perform simple decoding. If the separation of exploit and delivery becomes more widely adopted in bot code (as we anticipate it will), it suggests that NIDS could benefit greatly by incorporating rules that can detect follow-up connection attempts.

```
char encoder[]=
        "\xEB\x02\xEB\x05\xE8\xF9\xFF\xFF\xFF\x5B\x31\xC9\x66\xB9\xFF\xFF"
        "\x80\x73\x0E\xFF\x43\xE2\xF9";


  int xorkey=0x98;

  // Create local copies of the shellcode and encoder
  char *szShellCopy=(char*)malloc(iSCSize);
  memset(szShellCopy, 0, iSCSize); memcpy(szShellCopy, szOrigShell, iSCSize);
  char *szEncoderCopy=(char*)malloc(iEncoderSize);
  memset(szEncoderCopy, 0, iEncoderSize);
  memcpy(szEncoderCopy, encoder, iEncoderSize);

  if(pfnSC)
    pfnSC(szShellCopy, iSCSize);

  char *szShellBackup=(char*)malloc(iSCSize);
  memset(szShellBackup, 0, iSCSize);
  memcpy(szShellBackup, szShellCopy, iSCSize);

  // Set the content size in the encoder copy
  char *szShellLength=(char*)&iSCSize;
  szEncoderCopy[ENCODER_OFFSET_SIZE]=(char)szShellLength[0];
  szEncoderCopy[ENCODER_OFFSET_SIZE+1]=(char)szShellLength[1];

  // XOR the shellcode while it contains 0x5C, 0x00, 0x0A or 0x0D
  while(contains(szShellCopy, iSCSize, '\x5C') ||
        contains(szShellCopy, iSCSize, '\x00') || \
        contains(szShellCopy, iSCSize, '\x0A') ||
        contains(szShellCopy, iSCSize, '\x0D'))
    {
      memcpy(szShellCopy, szShellBackup, iSCSize); xorkey++;
      for(int i=0;i<iSCSize;i++) szShellCopy[i]=szShellCopy[i]^xorkey;
      szEncoderCopy[ENCODER_OFFSET_XORKEY]=xorkey;
    }

  free(szShellBackup);
```

Fig. 1.2. Agobot shell-encoding routine for malware delivery.

1.3.7 Obfuscation Mechanisms

Obfuscation refers to mechanisms that are used to hide the details of what is being transmitted through the network and what arrives for execution on end hosts. While none of the bots we evaluated included TCP obfuscations such as those described in [20], the aforementioned encoders provide obfuscation in a limited way. However, if the same key is used in each encoded delivery, then signatures could be generated quickly that would recognize a particular bit sequence. *Polymorphism* has been suggested as a means for evading signatures based on specific bit sequences by generating random encodings.

The only bot that currently supports any kind of polymorphism is Agobot. There are currently four different polymorphic encoding strategies that are supported: POLY_TYPE_XOR, POLY_TYPE_SWAP (swap consecutive bytes), POLY_TYPE_ROR (rotate right), POLY_TYPE_ROL (rotate left). While this code appears to function as advertised, thorough analysis of its capabilities is left for future work.

Implications: While polymorphic botnet delivery appears to be a reality, it is not yet widely available across bot families. As such, a concentrated focus on polymorphism by the network security community may not be warranted at this time. However, while the polymorphic routine packaged with Agobot is rather simplistic, it is conceivable that future botnets will have significantly support for polymorphism. As a result, anti-virus systems and NIDS will need to eventually develop mechanisms to account for this capability.

1.3.8 Deception Mechanisms

Deception refers to the mechanisms used to evade detection once a bot is installed on a target host. These mechanisms are also referred to as *rootkits*. Of the four bots we analyzed, only Agobot had elaborate deception mechanisms. These include (i) tests for debuggers such as OllyDebug, SoftIce and procdump, (ii) test for VMWare, (iii) killing anti-virus processes, and (iv) altering DNS entries of anti-virus software companies to point to localhost.

Implications: The elaborate deception strategy of Agobot some ways represents a merging of botnets with other forms of malware such as trojans and has several implications. First, honeynet monitors need to be aware of malware that specifically targets virtual machine environments. Second, it suggests the need for better tools for dynamic analysis of this malware since simply executing them in VMware or debuggers will provide false information. Finally, as these mechanisms improve, it is likely to become increasingly difficult to know that a system has been compromised, thereby complicating the task for host-based anti-virus and rootkit detection systems.

1.4 Conclusions

Continued improvements and diversification of malware are making the task of securing networks against attacks and intrusions increasingly difficult. The objective of our work is to expand the knowledge base for security research through systematic evaluation of malicious codebases. We advocate an approach that includes both static analysis of source code and dynamic profiling of executables. In this paper we take a first step in this process by presenting an evaluation of four instances of botnet source code. We selected botnet code as our initial focus due to its relatively recent emergence as one of the most lethal classes of Internet threats.

Overall, our source code evaluation highlights the sophistication and diverse capabilities of botnets. The details of our findings include descriptions of the primary functional components of botnets organized into seven categories. Some of the most important of findings within these categories include the diverse mechanisms for sensitive information gathering on compromised hosts, the effective mechanisms for remaining invisible once installed on a local host, and the relatively simple command and control systems that are currently used. While the IRC-based command and control systems remain an area that the network security community can potentially exploit for defensive purposes, it is likely that these systems will evolve toward

something like a peer-to-peer infrastructure in the near future (if they are not already doing so).

The results in this paper represent a first step in a much larger process of decomposing and documenting malware of all types. Ultimately, we anticipate that the resulting database will enable *proactive* network security. Our immediate next steps will be to begin the process of dynamic profiling of botnet executables using tools like IDA Pro [4] and by running the executables in our own laboratory environment. Beyond that, we plan to use the lessons learned from this study to begin an IRC monitoring effort at our university border router with the objective of developing new methods for identifying botnet communications. We also plan to expand our on-going honeynet measurement efforts to include botnet monitoring.

Acknowledgements

This work is supported in part by ARO grant DAAD19-02-1-0304 and NSF grant CNS-0347252. The second author was supported in part by a Lawrence H.Landweber NCR Graduate Fellowship. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the above government agencies or the U.S. Government.

References

1. F-Secure Corporation's Data Security Summary for 2004. http://www.f-secure.com/2004, 2004.
2. California Man Charged in Botnet Attacks. Reuters, November 2005.
3. Honeynet Scan of the Month 32. http://www.honeynet.org/scans/scan32/, 2005.
4. IDA Pro. http://www.datarescue.com, 2005.
5. Regmon. http://www.sysinternals.com, 2005.
6. SoftICE Driver Suite. http://www.compuware.com/products/driverstudio/softice.htm, 2005.
7. Sophos virus analyses. http://www.sophos.com/virusinfo/analyses, 2005.
8. C. Associates. GTBot1. http://www3.ca.com/securityadvisor/pest/pest.aspx?id=453073312, 1998.
9. M. Bailey, E. Cooke, F. Jahanian, J. Nazario, and D. Watson. The Internet Motion Sensor: A Distributed Blackhole Monitoring System. In *Proceedings of the Network and Distributed Security Symposium*, San Diego, CA, January 2005.
10. P. Barford. The Wisconsin Advanced Internet Laboratory. http://wail.cs.wisc.edu, 2005.
11. J. Canavan. The evolution of irc bots. In *Proceedings of Virus Bulletin Conference 2005*, October 2005.
12. E. Cooke, F. Jahanian, and D. McPherson. The zombie roundup: Understanding, detecting and disrupting botnets. In *Proceedings of Usenix Workshop on Stepts to Reducing Unwanted Traffic on the Internet (SRUTI '05)*, Cambridge, MA, July 2005.
13. P. Cousot, R. Cousot, J. Feret, L. Mauborgne, A. Mine, D. Monniaux, and X. Rival. The Astree Static Analyzer. http://www.astree.ens.fr, 2005.
14. Coverity. Coverity Prevent. http://www.coverity.com, 2005.
15. DETER. A laboratory for security research. http://www.isi.edu/deter, 2005.
16. D. Dietrich. Distributed Denial of Service (DDoS) Attacks/tools. http://staff.washington.edu/dittrich/misc/ddos/, 2005.

17. J. Evers. Dutch Police Nab Suspected Bot Herders. CNET News.com, October 2005.
18. German Honeynet Project. Tracking Botnets. http://]www.honeynet.org/papers/bots, 2005.
19. A. Gostev. Malware Evolution: January - March, 2005. http://www.viruslist.com, 2005.
20. M. Handley, C. Kreibich, and V. Paxson. Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics. In *Proceedings of the USENIX Security Symposium*, Washington, DC, August 2001.
21. The Honeynet Project. http://project.honeynet.org, 2003.
22. S. Kandula, D. Katabi, M. Jacob, and A. Berger. Botz-4-Sale: Surviving Organized DDos Attacks That Mimic Flash Crowds . In *Proceedings of the USENIX Symposium on Network Systems Design and Implementation*, Boston, MA, May 2005.
23. D. Kawamoto. Bots Slim Down to get Tough. CNET News.com, November 2005.
24. A. Kumar, V. Paxson, and N. Weaver. Exploiting underlying structure for detailed reconstruction of an internet scale event. In *Proceedings of ACM Internet Measurement Conference*, November 2002.
25. McAfee. W32-Spybot.worm. http//vil.nai.com/vil/content/v_100282.htm, 2003.
26. Metasploit. http://www.metasploit.com, 2005.
27. D. Moore, V. Paxson, S. Savage, C. Shannon, S. Staniford, and N. Weaver. Inside the slammer worm. In *Proceedings of IEEE Security and Privacy*, July 2003.
28. D. Moore and C. Shannon. The Spread of the Witty Worm. $http : // - www.caida.org/analysis/security/witty/$, 2004.
29. D. Moore, C. Shannon, and K. Claffy. Code red: A case study on the spread and victims of an internet worm. In *Proceedings of ACM Internet Measurement Workshop*, November 2002.
30. R. Pang, V. Yegneswaran, P. Barford, V. Paxson, and L. Peterson. Characteristics of internet background radiation. In *Proceedings of ACM Internet Measurement Conference*, Taormina, Italy, October 2004.
31. B. Saha and A. Gairola. Botnet: An Overivew. CERT-In White Paper, CIWP-2005-05, June 2005.
32. Sophos. Troj/Agobot-A. http//www.sophos.com/virusinfo/analyses/trojagobota.html, 2002.
33. Sophos. Troj/SDBot. http//www.sophos.com/virusinfo/analyses/trojsdbot.html, 2002.
34. S. Staniford, V. Paxson, and N. Weaver. How to Own the Internet in Your Spare Time. In *Proceedings of the 11th USENIX Security Symposium*, 2002.
35. I. Thomson. Hackers Fight to Create Worlds Largest Botnet. http://www.vnunet.com, August 2005.
36. J. Ullrich. Dshield. http://www.dshield.org, 2005.
37. D. Verton. Organized Crime Invades Cyberspace. http://www.computerworld.com, August 2004.
38. M. Vrable, J. Ma, J. Chen, D. Moore, E. Vandekieft, A. Snoeren, G. Voelker, and S. Savage. Scalability, fidelity and containment in the potemkin virtual honeyfarm. In *Proceedings of ACM Symposium on Operating Systems Principles (SOSP)*, Brighton, England, October 2005.
39. V. Yegneswaran, P. Barford, and D. Plonka. On the design and use of Internet sinks for network abuse monitoring. In *Proceedings of Recent Advances on Intrusion Detection*, Sophia, France, September 2004.
40. V. Yegneswaran, P. Barford, and J. Ullrich. Internet intrusions: Global characteristics and prevalence. In *Proceedings of ACM SIGMETRICS*, San Diego, CA, June 2003.