SRI International

CSL Technical Note • October 10, 2007

A Multi-perspective Analysis of the Storm (Peacomm) Worm

Phillip Porras and Hassen Saïdi and Vinod Yegneswaran
Computer Science Laboratory

`http://www.cyber-ta.og/pubs/StormWorm/`

Computer Science Laboratory•333 Ravenswood Ave.•Menlo Park, CA 94025•(650) 326-6200 •Facsimile: (650) 859-2844

A Multi-perspective Analysis of the Storm (Peacomm) Worm

Phillip Porras and Hassen Saïdi and Vinod Yegneswaran
Computer Science Laboratory

1 Introduction

Since early 2007 a new form of malware has made its presence known on the Internet by its prolific growth rate, its ability to distribute large volumes of spam, and its ability to avoid detection and eradication. Storm Worm (or W32.Peacomm, Nuwar, Tibs, Zhelatin), as it is known, is a highly prolific new generation of malware that has gained a significant foothold in unsuspecting Microsoft Windows computers across the Internet. Storm, like all bots, distinguishes itself from other forms of malware (viruses, Trojan horses, worms) by its ability to establish a control channel that allows its infected clients to operate as a coordinated collective, or botnet. However, even among botnets Storm has further distinguished itself by being among the first to introduce a fully P2P control channel, to utilize fast-flux [10] to hide its binary distribution points, and to aggressively defend itself from those who would seek to reverse engineer its logic.

The developers of Storm have retrofitted and improved its codebase over the last year, but its primary mission has remained to be a prolific propagator of spam (*e.g.*, a spambot). While the worm at its peak was deemed responsible for generating 99% of all spam messages seen by a large service provider [8], a reliable size estimate of the Storm botnet is hard to gauge because it uses a P2P communication protocol and there is no comprehensive measurement study to date. The only report that we are aware of is a conservative lower bound by Microsoft of over half a million infected machines of which it claims to have cleaned up over 267,000 machines infected with Storm. While certain other reports place Storm's botnet size at 50 million [14], most security experts consider this number to be unfounded and suspect a reasonable estimate to be between 1 million and 5 million.

Table 1: Summary of newsworthy Storm outbreaks (January through September 2007)

Date	Spam Tactic
Jan 17, 2007	European Storm Spam
April 12, 2007	Worm Alert Spam
June 27, 2007	E-card (applet.exe)
July 4, 2007	231st B-day
Sept 2, 2007	Labor Day (labor.exe)
Sept 5, 2007	Tor Proxy
Sept 10, 2007	NFL Tracker
Sept 17, 2007	Arcade Games

One might wonder why the Storm network has shown resilience in staying relevant and effective over an extended period of publicity (Table 1) that invariably would lead to its downfall. The effectiveness of Storm may be attributed to several factors that distinguish it from prior generations of malware:

- (a) Smart social engineering: Storm infection links are sent in emails that entice would-be victims by using highly topical and constantly changing spam campaigns, *e.g.,* subject lines about recent weather disasters [12], or holidays [4]
- (b) An ability to spread using client-side vulnerabilities: merely clicking on the wrong URL link from an unsolicited email may be enough to infect one's computer, and the apparent pool of users willing to do this may be in the millions
- (c) An ability to lure victims to malicious URLs by hijacking existing chat sessions [7]
- (d) An effectively obfuscated command and control (C&C) protocol overlaid on the Overnet P2P network
- (e) Actively updating the spambot client binaries to adapt to the latest OS upgrades, malware removal heuristics, and security patches

Despite all the hype and paranoia surrounding Storm, the inner workings of this botnet largely remain a mystery. Indeed, Storm is believed to have an automated distributed denial of service (DDoS) feature to dissuade reverse engineering, which gets triggered based on situational awareness gathered from its overlay network, *e.g.,* when the count of spurious probes crosses a certain threshold [3]. It has also been reported that these defenses have been turned on those that have posted their analysis results of Storm [13]. In this paper, we attempt to partially address voids in our collective understanding of Storm by providing a multi-perspective analysis of various Storm clients. Our analysis includes a static dissection of the malware binary and the characteristics of the Storm worm's network dialog as observed from multiple infection traces.

Finally, we do not only seek to analyze Storm for the greater understanding, but also to develop solutions that can help detect its presence, even as we expect Storm to continue to evolve and elude host security products. In this report we present our modifications to SRI's BotHunter Free botclient detection system (http://www.cyber-ta.org/BotHunter). We explain how BotHunter has been augmented to hunt for Storm infections, as well as other forms of spambot infections.

2 Static Analysis of Storm

Many variations of Storm have been released with each outbreak. Static analysis of the different released binaries provides an efficient way to discover logical similarities, as well as newly introduced changes to Storm's logic. We focus our analysis here on the version released on Labor day (September 2, 2007) in the form of the executable `labor.exe`.

2.1 Overview of `labor.exe` PE file

Along with other variants of Storm, `labor.exe` is packed using a custom packer employing known encryption routines. An elaborate discussion of the cryptographic code and rootkit techniques employed in Peacomm.C is provided in [1]. An initial investigation of the PE header (below) shows a legitimate entry point.

```
labor.exe:     file format efi-app-ia32
      architecture: i386, flags 0x0000012f:
      HAS_RELOC, EXEC_P, HAS_LINENO, HAS_DEBUG, HAS_LOCALS, D_PAGED
      start address 0x000000000040813a

      Sections:
      Idx Name          Size      VMA               LMA               File off  Algn
      0 .text         0000fe00  0000000000401000  0000000000401000  00000400  2**2
                      CONTENTS, ALLOC, LOAD, READONLY, CODE
      1 .rdata        00006a00  0000000000418000  0000000000418000  00010200  2**2
                      CONTENTS, ALLOC, LOAD, READONLY, DATA
      2 .data         00000600  0000000000421000  0000000000421000  00016c00  2**2
                      CONTENTS, ALLOC, LOAD, DATA
      3 .reloc        0000b1cd  0000000000422000  0000000000422000  00017200  2**2
                      CONTENTS, ALLOC, LOAD, CODE, DATA
```

When `labor.exe` starts at address 0x40813a, it executes an XOR decryption of a data bloc of size 0x9EA8. This first execution stage produces decrypted code starting at address 0x423200. This address is the beginning of the second execution stage in which the executable performs the following actions:

- Decrypts more code that constitutes the third and final stage of execution where the bulk of Storm's logic is executed.
- Creates a copy of itself called `spooldr.exe`
- Modifies the `tcpip.sys` driver
- Creates the `spooldr.sys` driver

Figure 1 shows the original code of `labor.exe`. A quick analysis of the code determines which instruction jumps to the newly decrypted code starting at address 0x423200. This instruction is highlighted in Figure 1. The created file `spooldr.exe` is an exact copy of the malware in its encrypted form, but the `spooldr.sys` driver can be created in either its encrypted or decrypted form, depending on the version of Storm (different versions of Storm might infect drivers other than the tcpip driver). Storm versions also differ in their implementations of checks to detect debuggers, virtual environments such as VMware and Virtual PC, that lead the code into an infinite loop whenever such environments are detected. The versions we have analyzed implement roughly the same logic, modulo the anti-debugging and anti-malware analysis techniques employed. Also, these different versions are created from a core common code base that is customized by the malware's author(s).

In what follows we will describe (i) how the infection results in a rootkit installation, (ii) how the malware is started after reboot once it infects a host, and (iii) a more in-depth analysis of the common code base that represents Storm's logic. We will illustrate along the way some of the difference between some versions of the malware.

```
.text:0040813A        public start
.text:0040813A start        proc near
.text:0040813A        mov     eax, 0FE79621h
.text:0040813F        sub     eax, 0FA56421h
.text:00408144        push    eax
.text:00408145        push    eax
.text:00408146        call    sub_40818C
.text:0040814B
.text:0040814B loc_40814B:        ; CODE XREF: .text:00408189
.text:0040814B        call    sub_40816D
.text:00408150        mov     eax, [ecx]
.text:00408152        add     eax, edx
.text:00408154        mov     edx, 0F12348h
.text:00408159        lea     esi, [esi+edx-0F12344h]
.text:00408160        mov     edi, esi
.text:00408162        add     eax, 12EE62Dh
.text:00408167        rol     eax, 2Fh
.text:0040816A        jmp     short loc_408180
.text:0040816A start        endp
.text:0040816A
.text:0040816C ;
.text:0040816C        retn
.text:0040816D
.text:0040816D ;  S U B R O U T I N E
.text:0040816D
.text:0040816D
.text:0040816D sub_40816D      proc near        ; CODE XREF: start:loc_40814B
.text:0040816D        mov     edx, 42D170h
.text:00408172        mov     edx, [edx]
.text:00408174        sub     eax, eax
.text:00408176        push    eax
.text:00408177        push    eax
.text:00408178        push    eax
.text:00408179        push    eax
.text:0040817A        push    eax
.text:0040817B        call    edx
.text:0040817D        mov     ecx, esi
.text:0040817F        retn
.text:0040817F sub_40816D      endp
.text:0040817F
.text:00408180 ;
.text:00408180
.text:00408180 loc_408180:        ; CODE XREF: start+30
.text:00408180        mov     edx, esp
.text:00408182        mov     esp, edi
.text:00408184        push    eax
.text:00408185        mov     esp, edx
.text:00408187        cmp     ebx, esi
.text:00408189        jg      short loc_40814B
.text:0040818B        retn
```

The ret instruction pops the computed address of the second phase execution

```
.text:0040818C
.text:0040818C ;  S U B R O U T I N E
.text:0040818C
.text:0040818C ; Attributes: bp-based frame
.text:0040818C
.text:0040818C sub_40818C      proc near        ; CODE XREF: start+C
.text:0040818C
.text:0040818C arg_0        = dword ptr  8
.text:0040818C
.text:0040818C        push    ebp
.text:0040818D        mov     ebp, esp
.text:0040818F        mov     esi, [esp+arg_0]
.text:00408193        mov     ebx, [esp+arg_0]
.text:00408197        add     ebx, 9EA8h
.text:0040819D        push    esi
.text:0040819E        leave
.text:0040819F        retn    4
.text:0040819F sub_40818C      endp
```

Figure 1: Storm’s Code

2.2 Drivers Infection and Rootkit Driver Install

The start function of the rootkit driver `spooldr.sys` is:

```
start         proc near
var_4       = dword ptr -4

              push ecx
              call sub_719C
              xor eax, eax
              push eax
              push offset sub_760D
              push eax
              push eax
              push eax
              push 1
              lea eax, [esp+1Ch+var_4]
              push eax
              call ds:PsCreateSystemThread
              push offset aSystemrootSyst ; "\\SystemRoot\\SYSTEM32\\ntoskrnl.exe"
              call sub_3E96
              push offset aSystemrootSy_0 ; "\\SystemRoot\\SYSTEM32\\drivers\\tcpip.sys"
              call sub_3E96
              xor eax, eax
              pop ecx ;
              call sub_760D
              retn 8
 start        endp
```

The first function executed by the driver is a subroutine sub_719C where `PsSetLoadImageNotifyRoutine` is called. This function routine registers a driver-supplied callback that is subsequently notified whenever an image is loaded for execution.[1]

```
sub_719C        proc near ; CODE XREF: start+1
                push   offset sub_40AF
                call    PsSetLoadImageNotifyRoutine
sub_719C        endp
```

The argument to `PsSetLoadImageNotifyRoutine` is a subroutine (sub_40AF) that terminates a list of processes and drivers running on the infected host *i.e.*, every time the machine is rebooted, the infected driver `tcpip.sys` will spawn the rootkit `spooldr.sys`, and every time a driver or a program in the undesired list is launched, it is immediately terminated. The following is the list of all executables and drivers that are targeted by termination:

Executables: Consult the complete executables that the malware disables (see the appendix to this document).

Drivers: `vsdatant.sys`, `watchdog.sys`, `bcfilter.sys`, `bcftdi.sys`, `bc_hassh_f.sys`, `bc_ip_f.sys`, `bc_ngn.sys`, `bc_pat_f.sys`, `bc_prt_f.sys`, `bc_tdi_f.sys`, `filtnt.sys`, `sandbox.sys`, `mpfirewall.sys`, `rdriv.sys`, `wincom32.sys`.

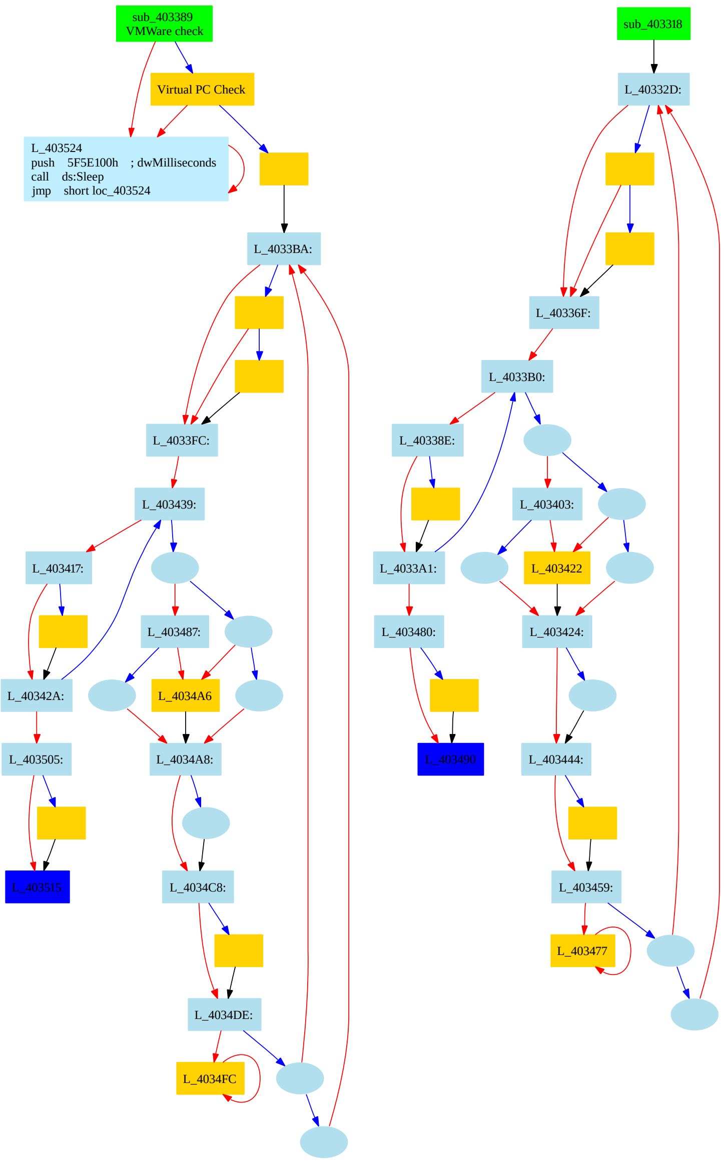
Some of the files in the undesirable list correspond to older versions of Storm executable files and rootkits. It has been suggested that the development of earlier versions of Storm were rushed [6], and the newer versions ensure that the buggy instances are cleaned up.

2.3 Understanding Storm's Logic

Once the initial phases of decryption and driver infection are achieved, a third phase of execution starts at a new OEP. The PE header's initial OEP of 0x40813a is no longer a valid entry point. When loading the unpacked version of `labor.exe`, IDA interprets 0x40813a as the OEP, and produces an assembly code that needs to be cleaned for further analysis. The second phase of execution indicates that the code jumps to a new OEP at address 0x40349F. The code at address 0x40349F looks like this:

```
push    0
call    sub_403318
retn
```

[1] http://msdn2.microsoft.com/en-us/library/ms802949.aspx



```
L_403524
push    5F5E100h     ; dwMilliseconds
call    ds:Sleep
jmp     short loc_403524
```

Figure 2: Difference between two versions of Storm. On the left we have applet.exe with VMware and Virtual PC checks. On the right we have labor.exe with no checks for virtual machine environments.

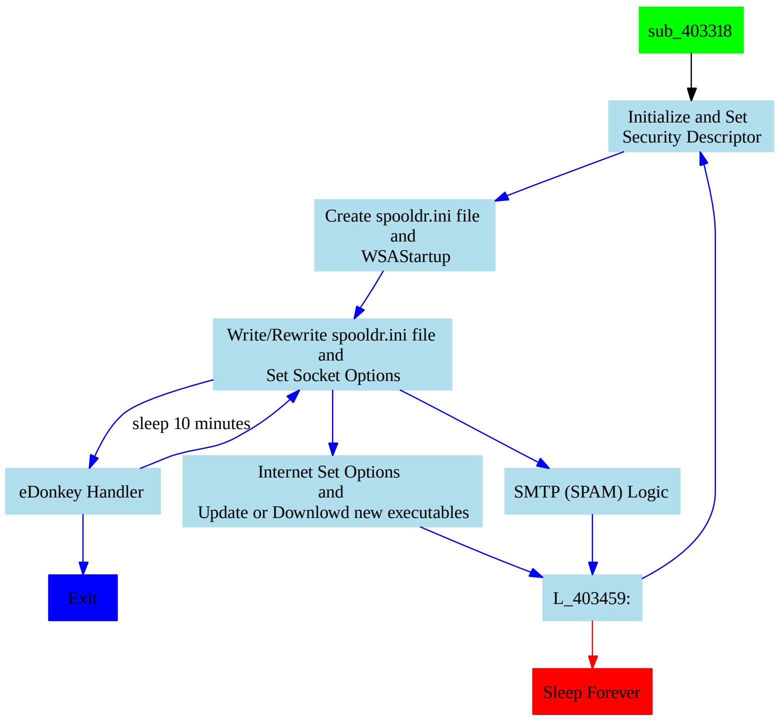

sub_403318
Initialize and Set Security Descriptor
Create spooldr.ini file and WSAStartup
Write/Rewrite spooldr.ini file and Set Socket Options
sleep 10 minutes
eDonkey Handler
Internet Set Options and Update or Downlowd new executables
SMTP (SPAM) Logic
Exit
L_403459:
Sleep Forever

Figure 3: Overview of Storm's Logic

We create a function at this address with the name start, and we identify function sub_403318 as the implementation of the core of Storm's logic. To understand Storm's logic, we need to generate a clean assembly that will allow us to build a control flow graph (CFG) of its code, recover all API calls, and identify their arguments. The first observation to note is that unlike other Storm variants, the main function sub_403318 in our version labor.exe does not start with some checks for virtual platforms such as VMware and Virtual PC.

To illustrate the differences between these two versions, we display in Figure 2 the control flow graphs of the two versions. The figure on the left side is applet.exe with VMware and Virtual PC checks. The figure on the right is `labor.exe` with no checks for virtual machine environments. Our static analysis tool-set allows us to quickly identify difference between different versions of malware and allows us to focus our attention on the key difference between versions. In subsequent subsections, we will explore the common functionality among the different versions of Storm that we analyzed. Newer versions of Storm seem to have dropped the checks for virtual environments often used by malware analyzers, in favor of encrypting the drivers that are created. This suggests that the malware's writers are far more interested in taking total control of infected hosts, hiding themselves from host monitoring software, and hiding the techniques that are employed to do so.

2.3.1 Storm Logic's Overview

Figure 3 illustrates a high-level annotation of the different blocks of Storm's code. Storm's code contains an initialization phase where the initialization file `spooldr.ini` is created and initialized, followed by a network initialization phase where Storm specifies the version of Windows Sockets required and retrieves details of the specific Windows Sockets implementation. Once the initialization phase is completed, the malware uses `spooldr.ini` as a seed list of hosts to contact for further coordination with infected peers. The coordination is achieved using the eDonkey/Overnet protocol. The malware retries to initiate such communication every ten minutes if no hosts in the initial list of peers are responsive. If some of the hosts are responsive, three main activities are triggered:

- Update the list of peers and store the new list in `spooldr.ini`.
- Initiate download of new spam templates or updates of existing executables.
- Initiate spamming and denial of service (Dos) activities.

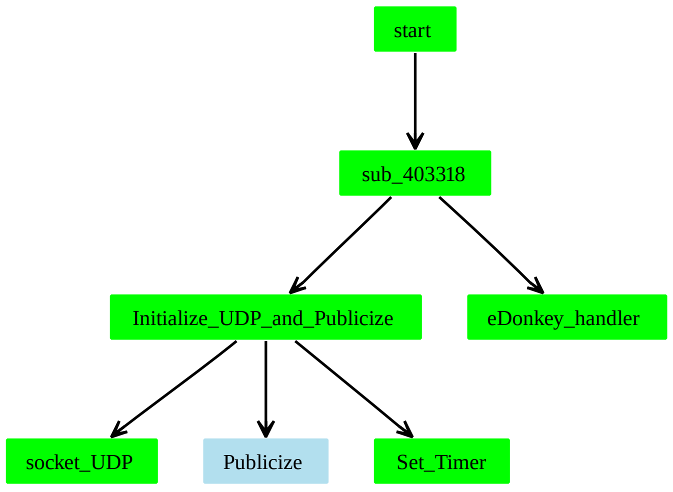


Figure 4: Overnet/eDonkey Protocol

The labeling of code blocks is achieved by first identifying all Windows API calls, their arguments, possible strings and numerical value references in each block, labeling each block by applying an ontology based on the ordering of API calls. This allows us to automatically identify the higher-level functionality of the malware instance such as networking activities and modifications to the local host. Based on the initial automated annotation, a more in-depth labeling is produced as in Figure3.

2.3.2 Initialization Phase

The initialization phase starts by creating a security descriptor for the file. This profile determines the level of access to the file. The descriptor is initialized with a null structure. Therefore, access is denied to the file so the process cannot be probed during execution. After the security descriptor initialization, the P2P component of the malware is initialized. A hard-coded list of 290 peers (number varies based on Strom version) shipped in the body of the malware is used to initialize the `spooldr.ini` file. Section 3 explains how the list of IP addresses of peers to contact is extracted from the `spooldr.ini` file format.

2.3.3 Overnet/eDonkey Communication Logic

Once the initial list of peers is established, the bulk of Storm's logic is executed using the Overnet/eDonkey protocol. A random list of peers is contacted by the infected host. If all communications do not result in an answer, the malware sleeps for 10 minutes and restarts the process of contacting its peers. The eDonkey protocol is executed in a block of instructions at address 0x004033B. It first initializes sockets to use the UDP protocol and issues a Publicize message to the peers it contacts.

```
loc_4033B0:                                   ; CODE XREF: sub_403318+74
                xor     bl, bl
                call    Initialize_UDP_and_Edonkey_PUBLICIZE        ; socket is called
                                                                    ; with argument 11h = 17
                                                                    ; for UDP protocol
                mov     esi, eax
                mov     eax, [esi]
                mov     ecx, esi
                call    Edonkey_CONNECT_SEARCH_and_PUBLISH          ; Respond to Publicize_ACK,
                                                                    ; Search, and Publish,
                                                                    ; and update spooldr.ini
                test    al, al
                jz      short loc_40338E
```

The control flow graph that corresponds to block 0x004033B is given in Figure 4. It shows how the first eDonkey communication initiated by the host is a Publicize command, followed by a call to the function edonkey_handler that manages incoming responses to the various eDonkey commands issued by the infected host. Our Static analysis of the eDonkey protocol implemented in Storm is correlated with the observed network traffic described in Section 3, in particular Figure 10 that illustrates the outbound traffic generated by Storm. Figure 5 shows the control flow graph of the eDonkey protocol handler and illustrates how Storm dialog sequences are generated. Our static analysis is correlated with the network analysis findings and corresponds to the observed traffic.

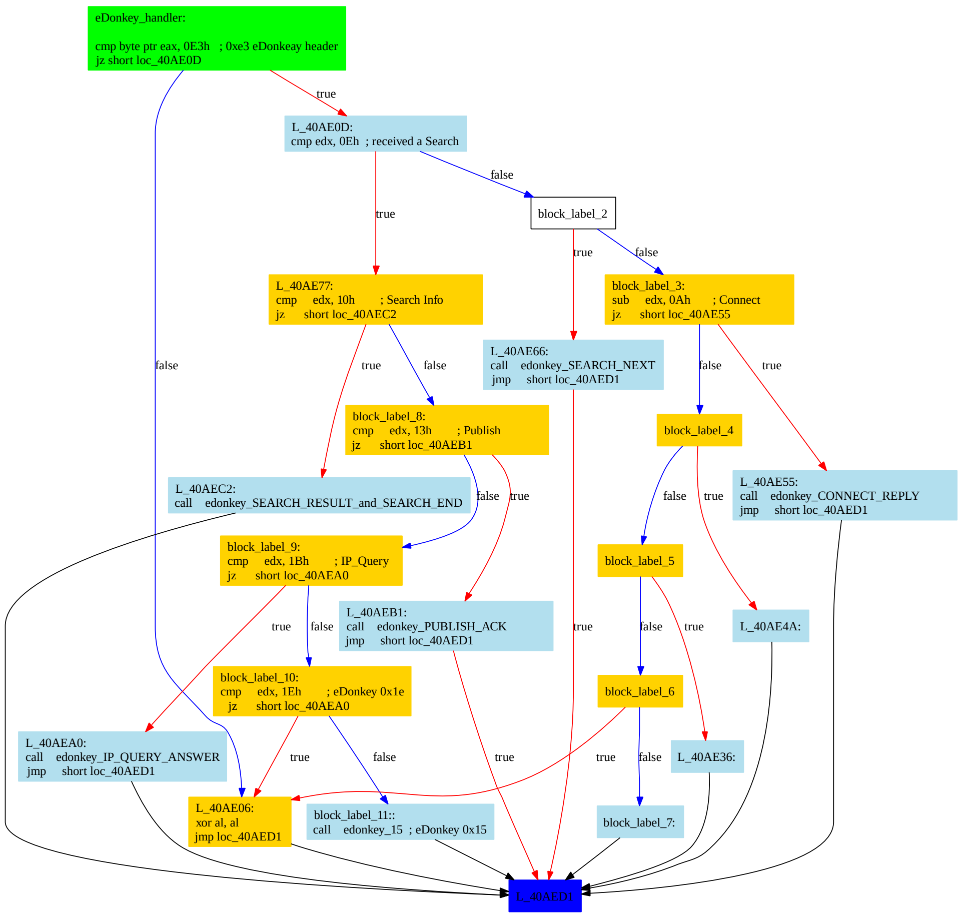


Figure 5: Overnet/eDonkey Protocol Handler

The interaction of Storm with its peers through the eDonkey protocol determines the next phase of execution of the malware. If the malware is unable to connect to the network or does not reach its peers, then it tries a connection every ten minutes. If a subset of the peers responds, then one of the following happens:

- Updates `spooldr.ini` with hashes of new peers;
- Downloads executables or updates existing executables;
- Scans the drives and collects email addresses and generates spam messages and DoS attacks.

2.3.4 Internet Download and Update

One particular dialog sequence of the eDonkey protocol results in a remote data retrieval of files that are downloaded on the infected host. We have identified the code that handles such downloads and describes its call graph in Figure 6. The malware writers seem to even have included entire utilities such as inflate.c from Zlib to handle downloaded compressed files.

2.3.5 Drive Scan

Storm has the ability to scan the drive of the infected computer to examine file content as shown in Figure 7. Files with the following extensions are scanned for their content: `.txt`, `.msg`, `.htm`, `.shtm`, `.stm`, `.xml`, `.dbx`, `.mbx`, `.mdx`, `.eml`, `.nch`, `.mmf`, `.ods`, `.cfg`, `.asp`, `.php`, `.pl`, `.wsh`, `.adb`, `.tbb`, `.sht`, `.xls`, `.oft`, `.uin`, `.cgi`, `.mht`, `.dhtm`, `.jsp`, `.dat`, and `.lst`.

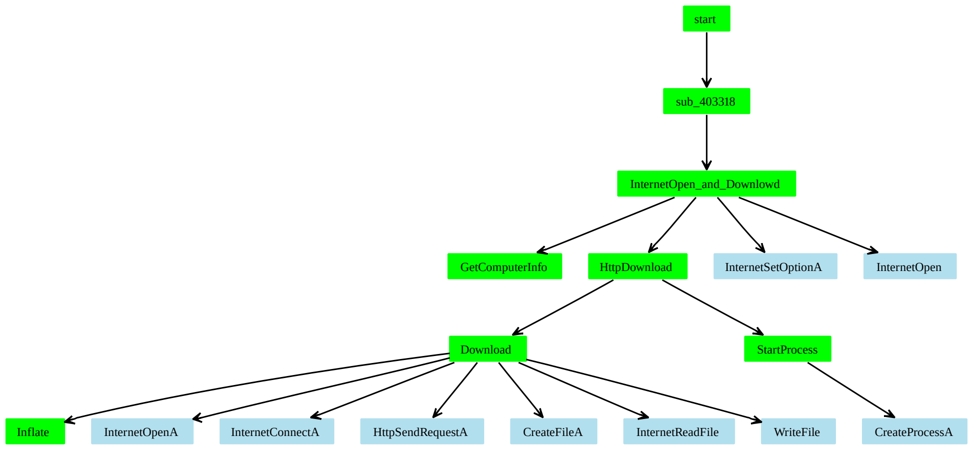


Figure 6: Internet Downloads and Updates Logic

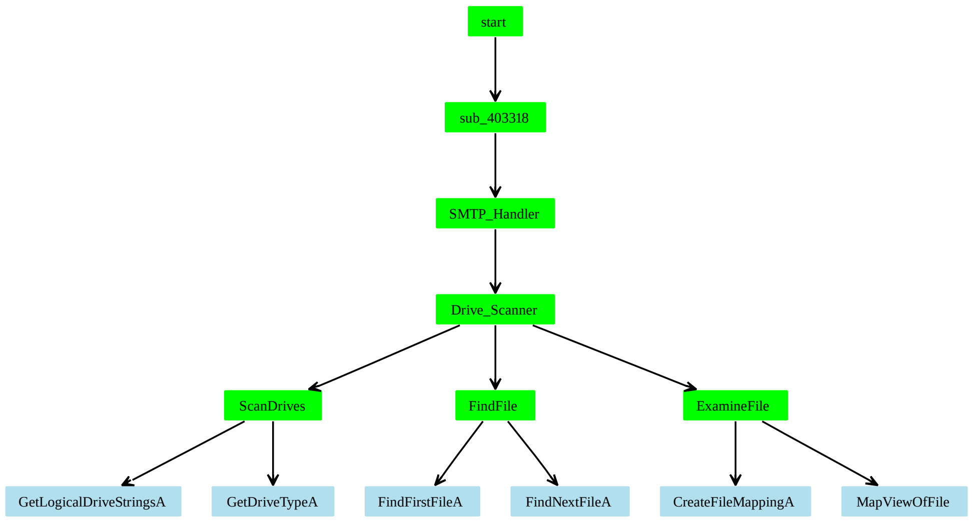


Figure 7: Drive Scans

Storm retrieves emails found in these files and gathers information about possible hosts, users, and mailing lists that are referenced in these files. In particular, it looks for strings like “yahoo.com”, “gmail.com”, “rating@”, “f-secur”, “news”, “update”, “anyone@”, “bugs@”, “contract@”, “feste”, “gold-certs@”, “help@”, “info@”, “nobody@”, “noone@”, “kasp”, “admin”, “icrosoft”, “support”, “ntivi”, “unix”, “bsd”, “linux”, “listserv”, “certific”, “sopho”, “@foo”, “@iana”, “free-av”, “@messagelab”, “winzip”, “google”, “winrar”, “samples” , “abuse”, “panda”, “cafee”, “spam”, “pgp”, “@avp.” , “noreply” , “local”, “root@”, and “postmaster@”.

3 Understanding Storm’s Network Dialog

We evaluated Storm’s network communications by monitoring the network communications of an infected system for over 7 hours. The host initiates communications by contacting the list of 290 hosts from spooldr.ini. The hosts in `spooldr.ini` are listed in the form `<hash>`=$<$ *ip* $><$ *port* $>$ 00 where `hash`, `ip`, and `port` are written in hexadecimal form. All 290 hosts are contacted within the first 30 seconds in three eDonkey Connect request packet bursts each lasting less than 2 seconds (contacting 27, 31, and 232 hosts respectively). A 10 second sleep is interspersed between the three episodes.

```
008052D5853A3B3D2A9B84190975BAFD=53855152054A00
004982069E5DB75721B54CFF33A26170=5955FC93123900
```

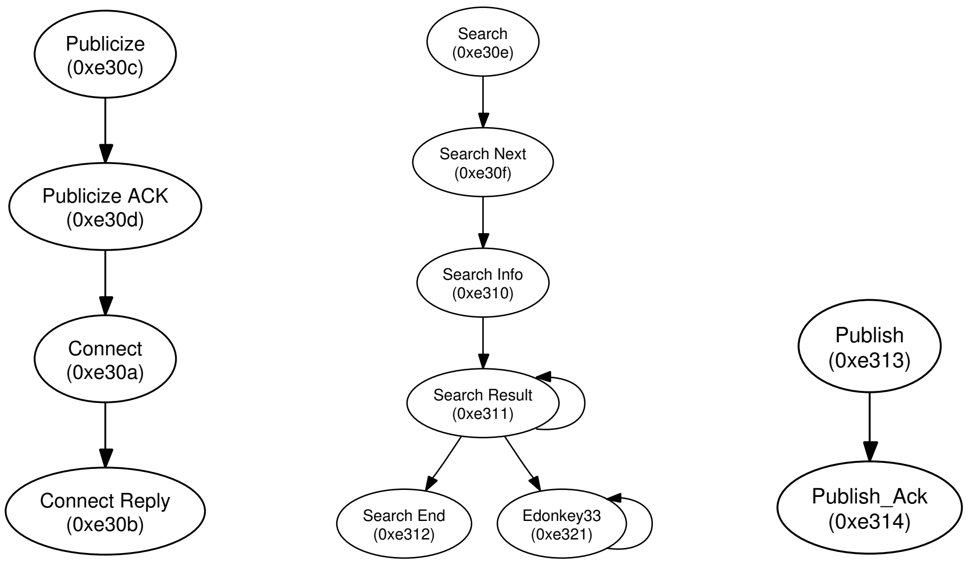


Figure 8: Recurring Storm Dialog Sequences

```
0042856B2ACE498B28D976190EA4F30C=443520D2410B00
0040A30E13C23842275F69AE7EFD59BA=C122902E4B4800
...
```

The network interactions of a Storm infected host are dominated by Overnet protocol communication which is used for its C&C and SMTP (TCP/25) communication which is used for sending spam. A Storm instance attaches itself to a variable high-order UDP port used for all Overnet communications distinguised by packets beginning with the byte sequences (0xe3). The next byte in the UDP packet indicates the message type. The Overnet messages observed could be classified in terms of the few recurring dialog sequences shown in Figure 8.

1. Overnet Control Plane Dialog: This dialog sequence is used to maintain the overnet topology and could have two or four states. Publicize (0xe30c) → Publicize ACK (0xe30d) → (optionally) Connect (0xe30a) → Connect Reply (0xe30b). The peers periodically use such messages to advertise themselves to other hosts and to probe their neighbor lists.

2. Overnet Data Plane Dialog: The data plane dialog could be grouped into two types of activities:

- *Data retrieval.* The typical data retrieval sequence in the Overnet protocol includes the following messages that are exchanged in order. Search (0xe30e) → Search Next' (0xe30f) → Search Info (0xe310) → Search Result' (0xe311) → Search End' (0xe312). The Overnet searching process proceeds in two phases. First, the client issues search requests on the hash of each keyword to which each peer responds with a Search Next message that suggests better peers to contact. Often, the peers also include themselves in that list, upon which the client issues a Search Info message advising the peer to restrict its search. The peer responds with one or more Search Result packets that provide the following information for each file that matches the keyword hash: filecontent hash and any meta data that describes the file (*e.g.*, name, size, content, format, availability) ending with a Search End message indicating that no more results are available. The client then chooses a suitable file and issues another round of data retrieval sequence requests on the filecontent hash to obtain a MetaTag that provides location information, *e.g.*, `bcp://ipaddr:port`.
- *Data publishing.* The data publishing sequence in the Overnet protocol involves just two messages, Publish (0xe13) → Publish Ack' (0x24). However, the actual publishing process is more involved. First, an MD4 hash of all keywords is computed and the metadata of the filecontent (including filename, filecontent hash, size) is published on each of these hashes. Second, the hash of the peer storing the file, along with the file URI (location), is published using hash of the filecontent as the key.

In our experiments with Storm, we noticed two deviant behaviors. First, in certain cases, remote peers followed Search Result packets with a series of unrecognized eDonkey messages type 0x21 (as opposed to Search Result) for which our Storm instance generates no response. We initially suspected that these might be related to Storm, but it turns this message type might correspond to "EDONKEY_MSG_MORE_RESULTS" messages, which is not parsed correctly by wireshark. We speculate that this nonresponsive behavior of Storm might be a means to distinguish Storm from other eDonkey clients.

Second, we also noticed a strange communication pattern, *i.e.,* Connect → Connect Reply' → IP Query → IP Query Reply' → TCP Connection attempt'. Notice the IP Query triggers outbound TCP connection attempts. These connection attempts never seem to succeed, suggesting that this is unexpected behavior. It turns out that this is actually part of the Overnet protocol's means to identify NAT firewalling. If a remote peer wants to identify whether its TCP ports are blocked, it executes this dialog sequence and waits for the inbound TCP connection attempt. In case of success, the remote peer responds with an IP Query End message.

Figure 9 provides a time volume graph of Storm's Overnet control plane (left) and data plane (middle, right) messages during a 7-hour infection trace. Figures 10 and 11 show the respective breakup of the outbound and inbound components. The graphs illustrate the following:

- Control plane messages *inbound eDonkey Connect/Connect Reply* dominate all eDonkey conversations. These message depict a decreasing trend over time, and most outbound eDonkey Connect requests are issued in the first 10 minutes.
- While all outbound Publicize messages are issued in the first 30 minutes, inbound Publicize message volumes remain steady during the entire infection period.
- Data plane messages are dominated by inbound Search requests. These messages are also initially bursty before stabilizing at an average rate of around 100 requests per minute.
- Of the remaining messages, unrecognized eDonkey type 33 messages dominate. These are often related to the eDonkey Search requests. We also see a steady rate of Publish and inbound IP Query activity. The Storm client faithfully responds to these message types, but does not generate them.

Figure 12 illustrates a time-volume graph of TCP packets, SMTP packets, spam messages, and smtp servers. Our analysis of this graph reveals the following findings. First, we find that except for the first 5 minutes almost all the TCP communication is dominated by spam. Second, we measured that hosts generate on average of 100 successful spam messages per 5 minutes, which translates to 1200 spam messages per hour or 28,800 messages per day. If we mutiply this by the estimated size for the Storm network (which we suspect varies between 1 million and 5 million, we derive that the total number of spam messages that could be generated by Storm is somewhere between 28 billion and 140 billon per day [2].

While such numbers might be mind-boggling they are inline with observed spam volumes in the Internet, *e.g.,* overall volume of spam messages in the Internet per day in 2006 was estimated to be around 140 billion [2]; Spamhaus claims to have been blocking over 50 billion spam messages per day in October 2006 [11], and AOL was blocking 1.5 billion spam messages per day in its network in June 2006 [5]. These numbers suggest that Storm could be responsible for anywhere between 17% and 50% of all spam that is generated on the Internet.

4 Detecting Storm P2P/Spambot Communication Patterns

The various forms of Storm P2P spambots are generally aggressive communicators, causing a locally infected host to significantly increase the volume and breadth of UDP and TCP communications to external targets. This increase in outbound communications is substantial, even beyond what typical scan-and-infect bots produce during their infection and coordination stages. Here we investigate the general question of detecting Storm-style bot infections from the perspective of network traffic analysis between a locally infected host and the Internet.

We have published a new free Internet release of BotHunter v 0.9.4 (http://www.cyber-ta.org/BotHunter) that is effective in tracking the broadest dialog patterns of spambots, and includes extensions to detect the Storm-specific overlay patterns exchanged among peer Storm bots. We have validated the extensions to our dialog model against several Storm spambot variants captured over the last several months. BotHunter is a dialog correlation system that tracks the two-way communication flows between internal assets and external entities [9] in search of locally infected systems. We have extended the BotHunter

[2]An important question might be whether all these hosts send spam simultaneously. Storm infected hosts constatly seem to send spam. However, they do so only when they are connected to the Internet. For completeness, the above number should be multiplied by the fraction of time Storm infected hosts are connected to the Internet per day.

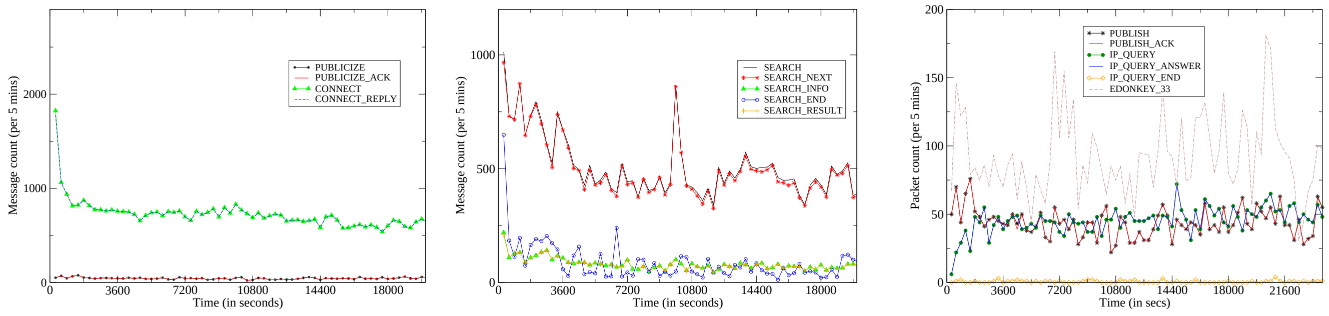


Figure 9: Time Volume Graph: Storm Dialog

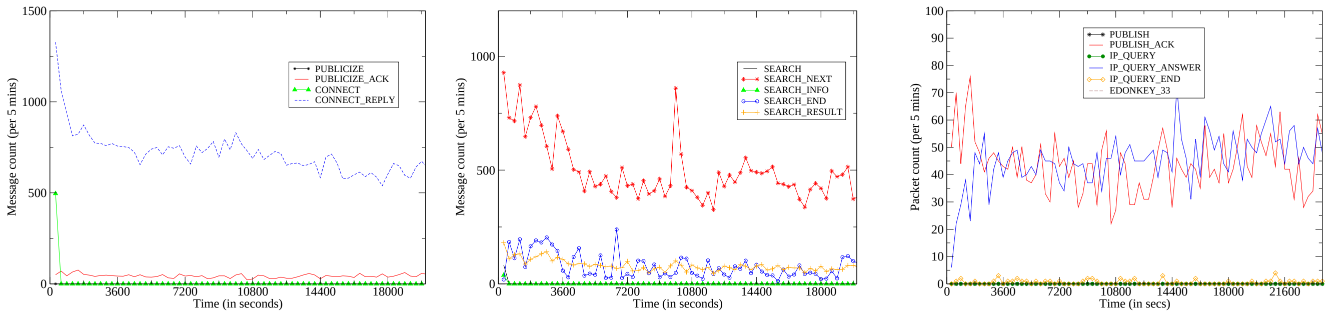


Figure 10: Time Volume Graph: Storm Outbound Dialog

dialog model to capture Storm peer coordination dialog and to identify spambot network transactions, including precursor DNS communication patterns.

Our goal here is to define a method for spambot identification that provides as much resiliency as possible to the rapid evolution of spambot logic, as demonstrated by the continual alterations to Storm's codebase over the last several months. BotHunter does not rely on single-packet content inspection as in traditional network IDS methods. Rather, BotHunter characterizes network communication flows as potential stages in an abstract malware infection life cycle, constructing an evidence trail from which it may conclude that a local asset is interacting with the Internet in a manner consistent with malware behavior. In [9] we present the details of our infection life cycle model and dialog correlation methodology. Here we extend BotHunter's infection life cycle model to address the unique communication patterns of peer-based botnet coordination strategies and spam propagation.

4.1 Analysis of the Storm Network Dialog Interaction Model

Figure 13 illustrates the BotHunter infection dialog life cycle model, and identifies what aspects of Storm's observed network communication patterns match the model. BotHunter constructs an evidence trail of network transactions, which we refer to as *dialog events*. We employ Snort (www.snort.org) to collect and map network transactions to the dialog events, which are then processed by BotHunter. Snort is augmented with a malware-specific scan detection plug-in that is used to identify inbound and outbound address scans associated with intrusion preparation, malware propagation, or spam distribution patterns.

Not all potential transactions in our model must be observed in order to declare a bot infection. In the case of Storm and its variants, at least two of the outbound dialog transaction sequences must be detected in order to cross a sufficient confidence threshold. The following is a summary of the dialog transaction events that compose the infection life cycle model in Figure 13:

1. **SCANNING EVENTS:** Applicable to scan-and-infect malware. This communication stage represents precursor activity by a potential attack source preparing for remote-to-local host infection. This stage is not applicable in spam-based bot propagation as found in Storm, as such bots do not acquire new victims through network address scanning.

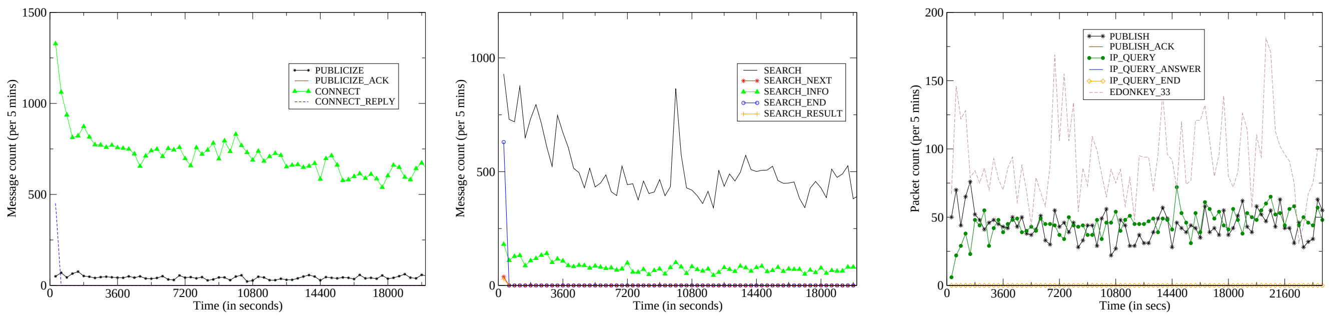


Figure 11: Time Volume Graph: Storm Inbound Dialog

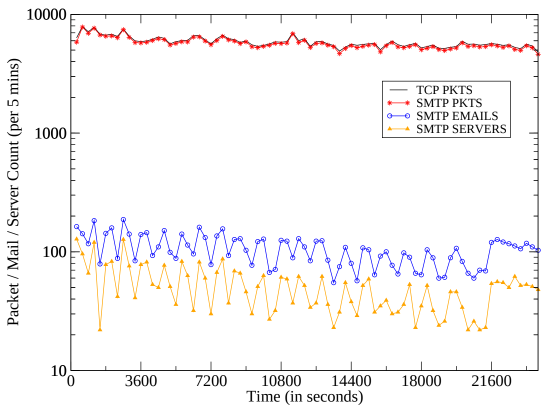


Figure 12: Time Volume Graph: TCP / SMTP Communication

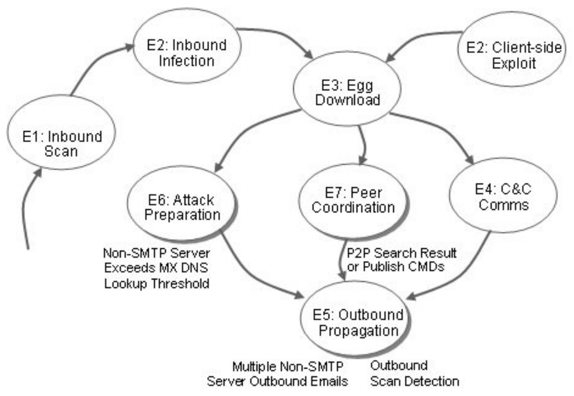


Figure 13: Dialog States of Storm

2. **EXPLOIT LAUNCH EVENTS:** Applicable to scan-and-infect malware. Here the internal victim host is attacked through a remote-to-local network communication channel. Storm and other spam bots propagate through email URL Link downloads and are then executed within the victim host.

3. **EGG DOWNLOAD EVENTS:** Applicable and detectable across malware families. Once infected, a compromised host is subverted to download and execute the full bot client codebase from a remote egg download site, usually from the attack source. However, in the case of Storm, this communication stage is observed over periods that are well delayed from the point of initial infection, sometimes many hours into the infection lifetime.

4. **COMMAND AND COORDINATION EVENTS:** Applicable to traditional C&C botnets. This communication stage is traditionally observed in botnets that support centralized C&C communication servers, such as IRC-based botnets. Storm peer-to-peer botnets utilize a peer-based coordination scheme.

5. **OUTBOUND ATTACK PROPAGATION EVENTS:** Applicable and detectable across all self-propagating malware families. This communication phase represents actions by the local host that indicate it is attempting to attack other systems or perform actions to propagate infection. In the case of spambots such as Storm, attack propagation can readily be discerned by the rapid and prolific communication of a non-SMTP-server local asset suddenly sending SMTP mail transactions to a wide range of external SMTP servers. In addition, spam and P2P bots both generate high rates of TCP and UDP connections to external addresses, often triggering intense streams of outbound port and IP address sweep dialog alarms.

   Example Outbound Attack Propagation Heuristics:

```
alert tcp !$SMTP_SERVERS any -> $EXTERNAL_NET 25 (msg:
   "BLEEDING-EDGE POLICY Outbound Multiple Non-SMTP Server Emails";
```

```
    flags: S,12; threshold: type threshold, track by_src,count 20,
    seconds 60; classtype: misc-activity; sid: 2000328; rev:7;)
```

Example SCADE and Snort OUTBOUND ATTACK PROPAGATION Alerts:

```
09/26/07-00:44:25.788068 [**] [1:2000328:7] E5[rb] BLEEDING-EDGE
  POLICY Outbound Multiple Non-SMTP Server Emails [**]
  [Classification: Misc activity] [Priority: 3] {TCP}
  192.168.1.55:3106 -> 65.54.244.200:25

09/26/07-00:48:12.173595 [**] [122:19:0] (portscan) UDP Portsweep
  [**] {PROTO255} 192.168.1.55 -> 83.221.199.217

09/26/07-00:47:30.283360 [**] [122:3:0] (portscan) TCP Portsweep
  [**] {PROTO255} 192.168.1.55 -> 200.142.128.82

09/26/07-00:49:16.946734 [**] [555:5555005:1] E5[sc] scade detected
  scanning of 30 IPs (fail ratio=0:37/22): 216.39.53.1:[25 ]
  81.25.173.124:[30242 ] 60.254.15.12:[8562 ] 88.204.209.110:[22149 ]
  85.90.196.22:[32344 ] 88.227.193.228:[26472 ] 65.54.244.200:[25 ]
  203.87.133.130:[23419 ] 68.218.184.188:[9566 ] 62.106.105.31:[5024]
  89.223.47.22:[12032 ] 203.177.237.144:[50041 ] [**] {TCP}
  192.168.1.55:0 -> 195.186.18.144:0
```

6. **LOCAL ASSET ATTACK PREPARATION EVENTS:** Applicable and detectable in spambot SMTP server list generation. This communication stage represents the locally infected victim performing actions that are indicative of preparing for attack propagation. For example, the collection of mail host IP addresses by a non-SMTP server local asset is a potential precursor action for spam distribution.

   Example Peer Coordination Alert:

```
alert udp !$SMTP_SERVERS any -> any 53 (msg:"E6[rb] BLEEDING-EDGE
  POLICY Possible Spambot -- Host DNS MX Query High Count"; content:
  "|01 00|"; offset: 2; depth: 4; content: "|00 0f 00 01|"; distance:
  8; threshold:type both, count 30, seconds 10, track by_src;
  classtype:bad-unknown; sid:2003330; rev:2;)
```

   Example Attack Preparation Alert:

```
09/26/07-00:44:28.802403 [**] [1:2003330:2] E6[rb] BLEEDING-EDGE
  POLICY Possible Spambot -- Host DNS MX Query High Count [**]
  [Classification: Potentially Bad Traffic] [Priority: 2] {UDP}
  192.168.1.55:1026 -> 192.168.1.3:53
```

7. **PEER COORDINATION EVENTS:** Applicable and detectable in P2P botnets. A P2P-based bot solicits and receives coordination instructions from a community of peers within the larger botnet. The protocol is used to synchronize bot actions and accept commands from a hidden controller. In Storm, peer coordination occurs via communications that are overlaid on the eDonkey UDP P2P protocol as discussed in Section 3. Here we apply BotHunter dialog warning heuristics to distinguish unique aspects of the Storm overlay communication dialog. The following rules capture unique aspects of Storm's use of eDonkey Search and Publish commands:

   Example Peer Coordination Heuristics (Snort Format):

```
alert udp $HOME_NET 1024:65535 -> $EXTERNAL_NET 1024:65535
  (msg:"E7[rb] BOTHUNTER Storm(Peacomm) Peer Coordination Event
  [SEARCH RESULT]"; content:"|E311|"; depth:5; rawbytes;
  pcre:"/[0-9]+\.mpg\;size\=[0-9]+/x"; rawbytes;
  classtype:bad-unknown; sid:9910013; rev:99;)

alert udp $HOME_NET 1024:65535 -> $EXTERNAL_NET 1024:65535
  (msg:"E7[rb] BOTHUNTER Storm Worm Peer Coordination Event
  [PUBLISH]"; content:"|E313|"; depth:5; rawbytes;
  pcre:"/[0-9]+\.mpg\;size\=[0-9]+/x"; rawbytes;
  classtype:bad-unknown; sid:9910011; rev:99;)
```

   Example Peer Coordination Alert:

```
09/26/07-00:48:48.375651 [**] [1:9910013:99] E7[rb] BOTHUNTER
  Storm (Peacomm) Peer Coordination Event 11 [**] [Classification:
  Potentially Bad Traffic] [Priority: 2] {UDP} 192.168.1.55:20176 ->
  72.24.115.141:49295
```

4.2 Example Storm(Peacomm) BotHunter Detetection Profile

BotHunter distinguishes successfully infected Storm spambot clients from the continual stream of background exploit chatter by searching for combinations of the above dialog events from each internal asset. The BotHunter correlator is not sequence-order dependent and tolerates dialog event omissions. It employs a weighted event threshold system to capture a minimum necessary and sufficient sequence of events that will permit bot profile declaration. For Storm and similar variants, any combination of two of the above dialog transaction events 5 through 7 is sufficient to declare a client infection.

Figure 14 presents an example BotHunter profile generated from a live instance of a Storm spambot infection (labor.exe). The header of the botProfile presents an overall confidence score, where a score of 0.8 is sufficient to declare a bot infection. The header also indicates the address of the internal victim and the attacking source. As Storm is a P2P-based botnet, BotHunter does not observe a C&C server, but rather lists the set of eDonkey clients that operate as spambot peers to our infected client during the bot detection window. Also included in the header is the set of addresses used as resources to prepare for the spambot propagation phase, in this case the DNS server used to collect an MX (email) server IP address list. Last, the header identifies the time windows in which the client botnet activity is observed.

Following the header, the profile includes details of the evidence trail used to detect the victim host infection. In this case, the Storm client manifests peer coordination activity, outbound spam proliferation, and an attack preparation phase in which it gathers a set of SMTP servers to whom it will propagate spam. Finally, the header includes the Unix tcpslice(1) command that may be used to isolate the bot communication evidence from a full network of traces.

The free Internet release of BotHunter, v0.9.4, is available for download at http://www.cyber-ta.org/BotHunter/. This software is designed for Fedora, SuSE, and Debian Linux systems.

5 Acknowledgements

We would like to thank Thorsten Holz and the incident handlers at the Internet Storm Center for the feedback and advice on this report.

References

[1] F. Boldewin. Peacomm.C Cracking the Nutshell. http://www.reconstructer.org, 2007.

[2] CommTouch. 2006 Spam Trends: Year of the Zombies. http://www.commtouch.com/documents/Commtouch_2006_Spam_Trends_-Year_of_the_Zombies.pdf.

[3] D. Danchev. Storm Worm's DDoS Attitude - Part two. http://ddanchev.blogspot.com/2007/09/storm-worms-ddos-attitude-part-two.html.

[4] DISOG. Latest Storm Worm Sharing Labor Day Greetings. http://www.disog.org/2007/09/latest-stormworm-filenames.html.

[5] B. Evans. It's time to take the spam fight to the bad guys. http://www.informationweek.com/story/showArticle.jhtml?articleID=16700428.

[6] E. Florio. The evolution of Peacomm to all in one trojan. http://www.symantec.com/enterprise/security_response/weblog/2007/04/-the_evolution_of_peacomm_to_al.html.

[7] E. Florio. MeSpam meets Zunker (and targets German users). http://www.symantec.com/enterprise/security_response/weblog/2007/05/-mespam_meets_zunker_and_target.html, 2007.

[8] S. Gaudin. Storm Worm Erupts Into Worst Virus Attack In 2 Years. http://www.informationweek.com/news/-showArticle.jhtml?articleID=201200849.

[9] G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee. Bothunter: Detecting malware infection through IDS-driven dialog correlation. In *16th USENIX Security Symposium (Security'07)*, 2007.

[10] Honeynet_Project. Know Your Enemy: Fast Flux Service Networks. http://www.honeynet.org/papers/ff/fast-flux.html, 2007.

[11] C. James. Spam hits the fan in spamhaus spat . http://www.computing.co.uk/vnunet/news/2166130/spam-hits-fan-spamhaus-legal.

[12] D. Kawamoto. Storm Worm rages across the globe. http://www.news.com/Storm-Worm-rages-across-the-globe/2100-7349_3-6151414.html.

[13] B. Schnier. Gathering Storm Superworm Poses Grave Threat to PC Nets. http://www.wired.com/politics/security/commentary/securitymatters/2007/10/securitymatters_1004, 2007.

```
Score:            1.5 (>= 0.8)
Infected Target:  192.168.1..55
Infector List:    <unobserved>
Egg Source List:  <unobserved>
C & C List:       <unobserved>
Peer Coord. List: 12.215.226.142 (6), 74.114.85.53 (5), 74.229.32.61 (6),
                   82.209.234.28 (4), 60.243.15.194 (9), 189.179.210.39 (4),
                   72.24.115.141 (5), 60.52.99.211 (4)
Resource List:    192.168.1.3 (2)
Observed Start:   09/26/2007 00:47:42.901 PDT
Report End:       09/26/2007 00:50:31.944 PDT
Gen. Time:        09/26/2007 00:51:05.983 PDT

INBOUND SCAN <unobserved>

EXPLOIT <unobserved>

EXPLOIT (slade) <unobserved>

EGG DOWNLOAD <unobserved>

C and C TRAFFIC <unobserved>

PEER COORDINATION
     12.215.226.142 (6) (00:50:23.882 PDT-00:50:23.892 PDT)
        event=1:9910013 (6) {udp} E7[rb] BOTHUNTER Storm Worm Peer Coordination
        Event 11 6: 20176->9051 (00:50:23.882 PDT-00:50:23.892 PDT)

     74.114.85.53 (5) (00:48:52.822 PDT-00:48:52.832 PDT)
        event=1:9910013 (5) {udp} E7[rb] BOTHUNTER Storm Worm Peer Coordination
        Event 11 5: 20176->60536 (00:48:52.822 PDT-00:48:52.832 PDT)
     :more...

OUTBOUND SCAN
     83.221.199.217 (00:48:12.173 PDT)
        event=122:19 {raw} (portscan) UDP Portsweep (00:48:12.173 PDT)

     81.228.11.100 (00:51:05.983 PDT)
        event=1:2000328 {tcp} E5[rb] BLEEDING-EDGE POLICY Outbound Multiple
        Non-SMTP Server Emails 3505->25 (00:51:05.983 PDT)

     205.234.240.99 (00:49:31.337 PDT)
        event=122:3 {raw} (portscan) TCP Portsweep (00:49:31.337 PDT)

     195.186.18.144 (00:49:16.946 PDT)
        event=555:5555005 {tcp} E5[sc] scade detected scanning of 30 IPs (fail
        ratio=0:37/22): 216.39.53.1:[25] 81.25.173.124:[30242] 60.254.15.12:
        [8562] 88.204.209.110:[22149] 85.90.196.22:[32344 ] 88.227.193.228:[26472]
        65.54.244.200:[25] 203.87.133.130:[23419] 68.218.184.188:[9566]
        62.106.105.31:[5024] 89.223.47.22:[12032] 203.177.237.144:[50041]
        0->0 (00:49:16.946 PDT)
     : more...

ATTACK PREP
     192.168.1.3 (2) (00:49:30.526 PDT) event=1:2003330 (2) {udp}
        E6[rb] BLEEDING-EDGE POLICY Possible Spambot -- Host DNS MX Query High Count
           3058->53 (00:49:30.526 PDT)
           3051->53 (00:50:26.897 PDT)

tcpslice 1190792862.901 1190793031.945 inputFile.tcpd | tcpdump -r - -w outputFile.tcpd 'host 192.168.1.55'
```

Figure 14: Example BotHunter Profile: Storm (Labor.exe)

[14] Wikipedia. Storm Worm. http://en.wikipedia.org/wiki/Storm_Worm.

6 Appendix

6.1 Executables Disabled by Storm

zclient.exe, msssrv.exe, mcshield.exe, fsbl.exe, avz.exe, avp.exe, avpm.exe, kav.exe, kavss.exe, kavsvc.exe, klswd.exe, ccapp.exe, ccevtmgr.exe, ccpxysvc.exe, iao.exe, issvc.exe, rtvscan.exe, savscan.exe, bdss.exe, bdmcon.exe, livesrv.exe, cclaw.exe, fsav32.exe, fsm32.exe, gcasserv.exe, icmon.exe, inetupd.exe, nod32krn.exe, nod32ra.exe, pavfnsvr.exe, 180ax.exe, 180sa.exe, 1ClickSpyClean.exe, a2antidialer.exe, a2pr.exe, aaupdt.exe, aawservice.exe, AceClubCasino.exe, acefilesearch.exe, aceziprun.exe, actalert.exe, ActiveNetworkMonitor.exe, adaware.exe, AdAway.exe, AdGold.exe, admagic.exe, Ad-PurgeDemo.exe, adsalert.exe, AdsCleaner.exe, adwarebazooka.exe, AdwareDeluxe.exe, AdwarePatrol.exe, adwarepunisher.exe, AdwareSpy4.exe, adwin.exe, a.exe, AgentSpyware.exe, AGSeiApp.exe, AGuardDogSuiteNT.exe, akl.exe, AKV.exe, alchem.exe, AlertSpy.exe, alevir.exe, AlfaCleaner.exe, alhlp.exe, alogcfg.exe, alsys.exe, answers.exe, antispam.exe, antispysoldier.exe, antivirusgolden.exe, AntivirusGolden.exe, apc_Admin.exe, App.exe, APS.exe, Armor2net.exe, AS100.exe, ashdisp.exe, ashmaisv.exe, ashserv.exe, ashwebsv.exe, aso.exe, aswupdsv.exe, atlantis.exe, atmclk.exe, AutoUpdateRun.exe, avgagent.exe, avgemc.exe, avkbar.exe, avsched32.exe, baigoo.exe, bargains.exe, BarMan.exe, BazookaBar.exe, bbchk.exe, bdmcon.exe, BearShare.exe, beta.exe, beyondremotefull.exe, bfk.exe, block-checker.exe, bpk.exe, BPSDataShredder.exe, BPSPopupShld.exe, BraveSentry.exe, cavrid.exe, cavtray.exe, ccevtmgr.exe, ccimscan.exe, cclaw.exe, cclgview.exe, ccpxysvc.exe, cfgwiz.exe, clamservice.exe, cpd.exe, cpf.exe, crypserv.exe, dfw.exe, dllhost32.exe, dsentry.exe, EbatesMoeMoneyMaker.exe, edonkey2000.exe, eitcwd.exe, ERS.exe, escorcher.exe, ETDScanner.exe, ethscout.exe, ETMP.exe, eww.exe, EyetideController.exe, farsighter.exe, FatBuster.exe, fdd.exe, ferret.exe, fie5344.exe, FireWalker.exe, FloboSpywareClean.exe, ForbesAlerts.exe, fpavupdm.exe, freedom.exe, freeprodtb.exe, FroggieScanDemo.exe, fs30.exe, f-sched.exe, fsdfwd.exe, fservice.exe, fsm32.exe, f-stopw.exe, ftviewer.exe, fvprotect.exe, fwnet64.exe, gcasdtserv.exe, GeoWhere.2.61.lite.exe, gestionnaireantidote.exe, GetByMail.exe, GiveMeToo.exe, Gnucleus.exe, GoodbyeSpy.exe, GrabBurn.exe, guard.exe, gv.exe, hackmon.exe, HbtOEAddOn.exe, hidownload.exe, HitVirus.exe, hwpe2.exe, IEWatch20.exe, inetupd.exe, install.exe, InternetSpy.exe, IntraKey.exe, irsetup.exe, isafe.exe, isamini.exe, isamonitor.exe, isass.exe, isclean.exe, ishost.exe, ismini.exe, isnotify.exe, issearch.exe, issvc.exe, itbill.exe, itunesmusic.exe, iwnvod.exe, JimmySurf.exe, JustRemoteITServer.exe, kav.exe, KeyLogger.exe, KeyLover21.exe, KillAndClean.exe, klpf.exe, klswd.exe, kpf4ss.exe, little_helper2.exe, livesrv.exe, LoggerConfigurator.exe, lsasrv.exe, lsass32.exe, magiclink.exe, MagPlayer.exe, MailSkinner.exe, Main.exe, MainWnd.exe, MalScr.exe, MalSwep.exe, MalwareDestroyer.exe, MalWhere.exe, mathchk.exe, mcagent.exe, mcshield.exe, mctskshd.exe, MemoryWatcher.exe, MNS.exe, MobMasher.exe, moni.exe, monifree.exe, MP3Galaxy.exe, MPPoker.exe, mscornet.exe, msecag.exe, msgsys.exe, MSHUTDOWN.exe, msls32.exe, MsnSniffer.exe, mssearchnet.exe, multipl.exe, mwsoemon.exe, MyVideoDaily2.exe, navapp.exe, navstub.exe, navw32.exe, NetCtl.exe, NetPumperIEProxy.exe, Netzip.exe, nisum.exe, Njexplor.exe, NLSupervisorPro.exe, no32mon.exe, nod32krn.exe, nod32ra.exe, nortonupdate.exe, nsmdtr.exe, nstask32.exe, nvctrl.exe, OemjiShare.exe, ofcdog.exe, optimize.exe, outpost.exe, Overseer.exe, OverSpy.exe, P2PNetworking.exe, pavfnsvr.exe, pbcpl.exe, PBOptions.exe, pcacmes.exe, PCagent.exe, PCBusted.exe, pcOrion.exe, pcps.exe, PCScanner.exe, PCSmokingGun2.exe, pctptt.exe, pcwatch.exe, PenguinPanic.exe, personalmoneytree.exe, pesttrap.exe, PestTrap.exe, PestWiper.exe, picx.exe, PKViewer.exe, plook.exe, pmmon.exe, pmsngr.exe, pmuninst.exe, powerscan.exe, ppmemcheck.exe, ppsys.exe, ppv5.exe, PrecisionTime.exe, PrivacyCrusaderDemo.exe, PrivateMailReader.exe, ProcAlert.exe, Pronto.exe, prt.exe, PSFree.exe, pxckdla.exe, qconsole.exe, qpanel.exe, rasautou.exe, RazeSpyware.exe, RCPAdmin.exe, Recorder.exe, regbar.exe, RegClean32.exe, RegistryCare.exe, RegistryFix.exe, RegistrySweeper.exe, regresc.exe, RemedyAntispy.exe, removeit.exe, RepSvc.exe, RFManager.exe, rpcsetup.exe, rtvscan.exe, RunBackGammon.exe, RunBingo.exe, Safewebsurfer.exe, sandboxieserver.exe, SAR.exe, SaveMyWork.exe, savscan.exe, sb32mon.exe, sbserv.exe, sbsse.exe, Scanner.exe, scanregw.exe, Scan&Repair2006.exe, Scrabble.exe, Sd2006.exe, SecCon.exe, SecretSpy.exe, SecurityiGuard.exe, SeeStat.exe, serv.exe, service32.exe, service.exe, SGFwSvc.exe, showbar.exe, ShowBehind.exe, sidefind.exe, SK60.exe, skin2000.exe, sks32proc.exe, SlimShield.exe, slman.exe, SmileySource.exe, smoke.exe, smpcpro.exe, smss32bk.exe, SnackMan.exe, sndsrvc.exe, Snoop.exe, SnowballWars.exe, Sp0.exe, spamihilator.exe, spampal.exe, spbbcsvc.exe, Spedia.exe, sp_rsser.exe, SpyAOL.exe, SpyBro.exe, spycl4.exe, SpyCleanerGold.exe, SpyCleanerPlatinum.exe, SpyFighter.exe, SpyGraphica.exe, SpyHeal.exe, SpyHunter.exe, SpyiBlock.exe, Spyinator.exe, SpyKiller.exe, SpyLax.exe, SpyMon.exe, SpyOnThis.exe, SpyPry.exe, SpyReaperProDemo.exe, spyrem.exe, spyshield.exe, SpySniper.exe, SpySpotter.exe, SpySub.exe, Spytector.exe, spytrooper.exe, SpyTrooper.exe, SpyViperProDemo.exe, Spyware_Annihilator.exe, SpywareBot.exe, SpywareDetector.exe, SpywareDisinfector.exe, SpywareQuake.exe, spywareremovalwizard.exe, SpywareRemover.exe, SpywareSlayer.exe, SpywareStormer.exe, SSDemo.exe, sservice.exe, Ssk.exe, ssp.exe, sss.exe, StaffCop.exe, stardialer.exe, StartPoker.exe, stinger.exe, STMonitor.exe, story.exe, sunshinebingo.exe, Surfkeeper.exe, svcmon.exe, sv.exe, swatcher.exe, swdoctor.exe, swnxt.exe, symwsc.exe, syscfg32.exe, sysd.exe, sysformat.exe, syslog.exe, Syslogin.exe, sysmgr32.exe, sysmgr64.exe, system.exe, taskdir.exe, tasker.exe, titanshield.exe, tmoagent.exe, ToolKeylogger.exe, TopSearch.exe, tpcl.exe, truedownloader.exe, TrustCleaner.exe, TTBSETUP.exe, TVS_B.exe, TWAB5.exe, u88.exe, UDC2006.exe, uert.exe, UltraKeyboard.exe, UnSpyPC.exe, !update.exe, updsvc.exe, userinit32.exe, usrprmpt.exe, USYP.exe, UTviewer.exe, VCatch.exe, vetmsg9x.exe, vetmsg.exe, vettray.exe, viewer.exe, view.exe, VIRTUESCOPE.exe, VirusRescue.exe, vptray.exe, was6.exe, wcantispy.exe, weather.exe, Weather.exe, webrebates.exe, websnitch.exe, wfdmgr.exe, whspeedrank.exe, WICleaner.exe, win16dll.exe, WinAV.exe, wincp.exe, windll.exe, winlogin.exe, winlogons.exe, winlogonsys.exe, WinPass.exe, WinSL.exe, winsrv32.exe, wmsmod32.exe, wnames.exe, wnetmgr.exe, words.exe, WorldAntiSpy.exe, wrclock.exe, ws.exe, wslogger.exe, WSMDI.exe, WTRTrial.exe, wupdt.exe, xcommsvr.exe, X-ConSpywareDestroyer.exe, xfr.exe, Xolox.exe, xp-antispy.exe, xSpyware.exe, ZangoAstrology.exe, zango.exe, ZangoTVTimes.exe, zapspot.exe, zcodec.exe, ZComService.exe, zilla.exe, ZipItFast.exe