UNIVERSITY OF AMSTERDAM

Faculty of Science
System and Network Engineering

Detection of peer-to-peer botnets

by

Matthew STEGGINK and Igor IDZIEJCZAK

UvA supervisor: Dr. Ir. C. DE LAAT
SURFnet supervisor: Ir. R. SPOOR
SURFnet supervisor: Dr. W. BIEMOLT

Research Project 1
Master of Science Program

Academic year 2007–2008

Acknowledgement

Amsterdam, February 4, 2008

We would like to express a word of gratitude for the opportunity, the University of Amsterdam and SURFnet gave us, to conduct our research project. We want to thank our supervisors Ir. R. Spoor and Dr. W. Biemolt for their time and effort guiding us through the project. They were always willing to help us and gave us the possibility to use SURFnet's infrastructure. Without their aid, we couldn't have completed our research. Also a word of thanks to the following persons for supplying us with files, information and advice:
Nicolas Albright (Disog.org), Dr. Chato H. Flores and Dr. Jose Nazario (Arbor Networks).

We gave SURFnet and the University of Amsterdam a deeper insight in the detection of Peacomm and decentralized peer-to-peer botnets in general. SURFnet can use this knowledge to successfully identify the threat of P2P botnets, now and in the future.

We gave our vision on the future and the technological possibilities these botnets have. These botnets are growing each day, and it is very scary what a botnet herder could do with all the bandwidth. As their possibilities seem endless, we don't think that the Internet society is ready yet to face these malicious acts but we are definitely going in the right direction.
All in all, it was a very educational experience and we have learned a lot in a very short time span!

Matthew Steggink and Igor Idziejczak

Research report for System and Network Engineering, University of Amsterdam, the Netherlands for the 2007–2008 course *Research Project 1* in cooperation with SURFnet.

Matthew Steggink <matthew.steggink@os3.nl>,
Igor Idziejczak <igor.idziejczak@os3.nl>.

Version 1.0.0 and compiled with LaTeX.

Abstract

Peer-to-peer botnets are becoming more sophisticated and more dangerous each day. The technical possibilities recently being used by these botnets indicate a more sophisticated peer-to-peer approach by using an embedded decentralized architecture. *"Traditional"* botnets could be easily detected with NetFlow [4]. As of January 2007, Peacomm saw daylight using a completely decentralized P2P protocol based on the Kademlia algorithm. Detection of decentralized architectures is very hard because it doesn't have a single point of failure. This is the reason SURFnet wanted to reevaluate their peer-to-peer botnet detection methods. Our research project focuses on how the *"new generation"* peer-to-peer botnets can be detected, in particularly Peacomm.

This document gives an introduction in botnets and describes the different architectures used by botnets. We explain the techniques used by botnets to evade detection. A case study of Peacomm with a network analysis can be found in this document. We have researched how Peacomm spreads, how it operates and how it hides itself. It also describes the details about detecting Peacomm. We have described characteristics which may help administrators to detect decentralized peer-to-peer botnet.

This document is especially intended for network administrators and people who are interest to know more about Peacomm and P2P botnets.

Contents

List of Figures

List of Tables

1 Introduction

One of the most significant threats to the Internet today is the threat of botnets. These are networks consisting of large numbers of computers engaging in malicious activities. For the most part, this belief has been supported by the conjecture that at any moment in time, there is a large collection of well-connected compromised machines that can be coordinated to take part in malicious activities [40].

As the *"traditional"* botnets are being shutdown (as IRC botnets can be easily detected because they have a single point of failure), the newer and more dangerous botnets are moving to more resilient architectures. The newer generation botnets are also using new techniques to hide their botnet and their tracks and are using a more sophisticated encryption method. This will make them ever harder to track and fight [42].

SURFnet knows this threat and already investigated if P2P botnets can be detected with NetFlow[4]. These were the first generation P2P botnets and more details can be found in the work by 2 fellow students, titled *Detecting peer-to-peer botnets* [24]. At that moment, no changes were needed in the way SURFnet detects botnets because they all had some sort of single point of failure.

The uprise of Peacomm introduces a new problem with a complete decentralized P2P protocol. Existing botnet detection techniques, such as NetFlow, aren't working as good as with *"traditional"* botnets. We have to investigate how these peer-to-peer botnets can be detected.

At the moment of writing, existing botnet families incorporate P2P techniques but are already covered by previous work [24]. Only one bot, Peacomm is build from the ground with a complete decentralized P2P structure. Peacomm is also known as Storm, Stormworm, CME711 and Nuwar but in our document we always refer to it as Peacomm. Our research will mainly focus on this bot with a case study of Peacomm.D, the most recent version of Peacomm. We will also conclude with a more general advice on how to detect this malicious traffic now and in the future as this kind of botnet families will arise in the future.

This research includes a Peacomm case study and investigates how this family of *"new generation"* botnets can be detected. This research is a part of the Master of Science programme System and Network Engineering (SNE) at the University of Amsterdam.

2 Background

2.1 Botnet

Malicious botnets are networks consisting of large numbers of *bots*. Bot is actually short for robot. Symantec [39] defines a bot as:

> Bots are similar to worms and Trojans, but earn their unique name by performing a wide variety of automated tasks on behalf of their master.

Bots are also called *"zombies"* because a computer (infected with a bot) performs a task given by its master. A botnet herder (also called a botnet master) is a person or a group , who control the whole botnet: they can give instructions or upload data to the botnet. Botnets are used to perform a wide variety of tasks but some of the most popular ones are sending spam or coordinating a DDoS [1] attack.

There are different ways to infect a computer with a bot (see table 1 for an overview). Often it searches the Internet to look for "unprotected" computers to infect by exploiting known software vulnerabilities but it can also being sent through email, or hidden in another program or installed by a malicious website, this software is also known as *malware*.

A peer-to-peer botnet incorporates a P2P protocol in his code. Although such a protocol already existed for a couple of years it is but recently that we have encountered this in botnets, see section 2.3 on page 10. Some peer-to-peer bots have used existing peer-to-peer protocols while others have developed custom protocols [14].

Threat	Description
Email	Malware in email messages
Exploit	To exploit a particular vulnerability in a system
File shares	Network shares provide a transport mechanism for malware
Instant messaging	Through sharing files with other users malware can spread
Dictionary attack	Guessing a user's password to get access
Internet downloads	Downloaded directly from Internet Web sites
Removable media	Malware can spread when using this media on different computers
Network scanning	Scan for computers that have open ports or attack randomly IP addresses
P2P networks	Install malware after downloading innocent looking files

Table 1: Malware threats (Source: Microsoft.com [1])

2.2 Botnet architectures

There are a few topologies described in this chapter. They differ from each other in how big they can get, how easy they can be detected and disrupted and how they exchange messages and commands (command and control).

2.2.1 Centralized

The oldest type of topology is the centralized form, see figure 1 on the following page. There is one central point that forwards commands and data between the botnet herder and his botnet clients. The big advantage is that there is little latency. The biggest drawback is that they can be more easily detected then decentralized, since all the connections lead to a few nodes. Also when a IRC server gets disrupted or disconnected, an entire branch or perhaps the whole botnet

[1] Distributed Denial of Service (DDoS) attack

could be taken offline because the botnet herder can not pass any messages to the bots anymore. Passing commands and data in centralized systems go through a central point: in most cases

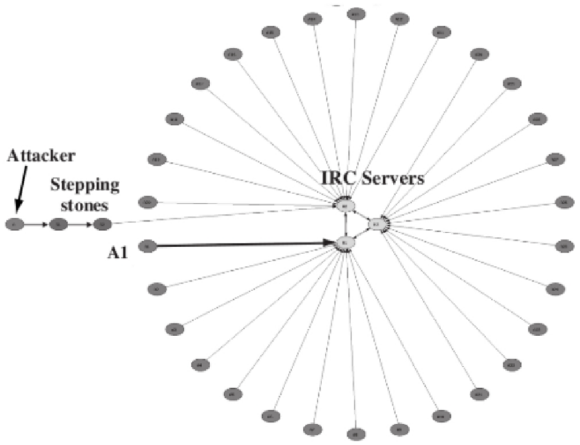

Figure 1: Centralized topology (source: Dave Dittrich et al. [15])

they all connect to a central C&C node or a central IRC server where they will receive their commands.[27] These IRC servers can be setup on hacked computers, or the botnets could use public IRC servers. When the IRC server has been taken offline, the botnet will become "lost", meaning that the botnet herder will not be able to send commands and data to the bots, rendering his bots useless. The more modern IRC botnets are more resilient because they use a list of IP addresses of alternate IRC servers, which they will use in case an IRC server has been taken offline.

2.2.2 Decentralized

A peer-to-peer (or P2P) computer network uses a partial or full mesh topology and the cumulative bandwidth of network participants. Peer-to-peer networks are typically used for connecting a very large number of nodes via ad hoc connections. These networks differ from the traditional client-server model because the peer nodes (both client and server) are all equal [12]. The decentralized topologies do not have one central node which can just be taken down to disable the botnet. Instead, they are all connected to eachother in a way. Some protocols use centralized servers for search operations (like napster [18] or define *"superpeers"* for maintaining an index or acting as a broker [2] [37]).

The peer-to-peer topology is fairly new, see figure 2 on the next page. It has some big advantages over the centralized topology. Peer-to-peer communication is harder to detect because they do not connect to a central node. When disconnecting bots from the peer-to-peer system, this will not have a significant effect on the remaining bots, making it harder to disrupt the whole botnet. The bots do require bootstrapping before they can join the botnet: they have to know an already connected bot, and then join the botnet through that bot. The connected bot will then pass information about the botnet to the new connected bot, effectively making it part of the botnet. There are a few implementations available for this topology to pass commands and data. One is to structure the botnet by using a location service using DHT such as chord [27] [7], but a drawback is that it reveals information about other nodes. Another method is using DHT with the Kademlia algorithm [11] (for an explanation, see section 3.1 on page 14). Peacomm uses the Overnet protocol, which is based on the Kademlia algorithm to communicate.

[2] Brokers can be used for i.e. registering servers and making them known to other nodes

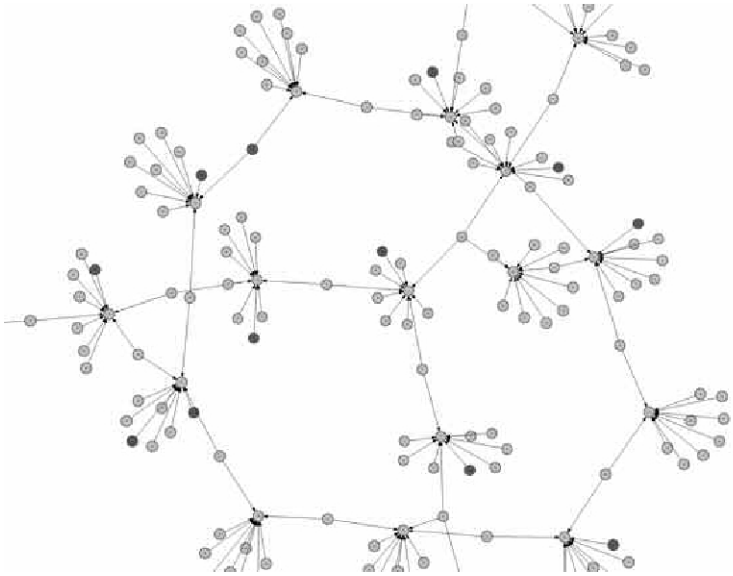

Figure 2: Peer-to-peer network (source: Dave Dittrich et al. [15])

2.2.3 Hybrid

Hybrid P2P systems [45] are not implemented and is theory only, see figure 3. There are a few distinct differences with the peer-to-peer and centralized topology. First, the bot communicates via a peerlist contained in each bot. Each bot has it's own fixed and limited list, which it will not reveal to other bots. When a bot is compromised, only a limited number of nodes are exposed. This way, the botnet does not require bootstrapping, it can connect to the botnet directly. There is a disadvantage about this method: only bots with a static IP can be chosen to appear in the list because it's hard coded. Each servant bot listens on a self determined port for incoming connections and uses a self-generated symmetric encryption key for incoming traffic. This makes it very hard to detect because each bots traffic uses a different encryption key [45]. The botnet master issues commands to his servant bots (listed in the peerlist). These servant will forward the commands to the other nodes.

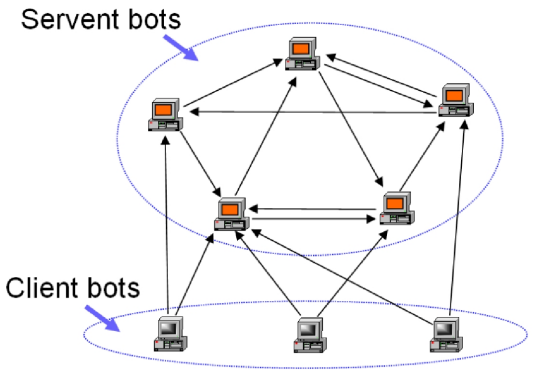

Figure 3: Hybrid peer-to-peer botnet (source: Ping Wang et al. [45])

2.3 History

In table 2, we can see the rise of P2P and P2P botnets. Actually, the use of P2P in malware started out in 1999 with the Samhain project [43]. In 2002, the Linux Slapper worm claimed to use a P2P algorithm in his source code [32] that could support 16 million peers. This P2P mechanism was never tested because its propagation was too noisy.

Date	Name	Type	Remarks
05/1999	*Napster*	P2P	First widely used central P2P service
03/2000	*Gnutella*	P2P	First decentralized P2P service
09/2000	*eDonkey*	P2P	Used checksum directory lookup for files
03/2001	*FastTrack*	P2P	Use of supernodes within the P2P architecture
07/2001	*BitTorrent*	P2P	Uses bandwidth currency for faster downloads
05/2003	*WASTE*	P2P	Small VPN-style network with RSA public keys
08/2003	*Agobot*	Bot	Rudimentary P2P mechanism analyzed by Dittrich et. al [15]
09/2003	*Sinit*	Bot	P2P bot using random scanning to find peers
11/2003	*Kademlia*	P2P	Uses DHT for decentralized architecture
03/2004	*Phatbot*	Bot	P2P bot based on WASTE
03/2006	*SpamThru*	Bot	P2P bot using custom protocol for backup
04/2006	*Nugache*	Bot	Peer-to-peer bot connecting to predefined peers
01/2007	*Peacomm*	Bot	Peer-to-peer bot based on Kademlia

Table 2: Timeline peer-to-peer protocols and bots [14].

Since the discovery of Peacomm a lot of updates have been made in the source code of these bots. On October 31, 2007, Symantec discovered Trojan.Peacomm.D [3], another variant of the peacomm worm. In spite of some minor changes, it is still the major version used today.

2.4 Evasion tactics

Botnets are commonly used for criminal activities like a distributed denial-of-service (DDoS), sending spam or setting up phishing sites. Because of the use of botnets for criminal activities, they want to cover up their tracks. Botnet herders use different techniques to hide their botnet to prevent them being detected. Also they use techniques to make their command and control bots more redundant, in case one gets shutdown. Through the years these botnets have become more and more sophisticated and harder to track, but however they can still be tracked.

2.4.1 Fast-flux DNS

Fast-flux DNS is a technique which overloads the A records in the DNS server. One A record will have multiple IP addresses, making it redundant: each client will try the IP addresses, until it can succesfully connect to one[3] [9].
See the example below, the A record of this domain has multiple IP addresses.

```
;; ANSWER SECTION:
windowsoxonline.com.    300     IN      A       58.103.16.179
windowsoxonline.com.    300     IN      A       59.10.217.239
windowsoxonline.com.    300     IN      A       59.17.208.96
windowsoxonline.com.    300     IN      A       61.76.250.122
windowsoxonline.com.    300     IN      A       61.238.72.41
```

[3] http://www.symantec.com/security_response/writeup.jsp?docid=2007-103120-0804-99

```
windowsoxonline.com.    300     IN      A       69.228.33.128

...
...
;; AUTHORITY SECTION:
windowsoxonline.com.    300     IN      NS      ns0.fyukbz.com.
windowsoxonline.com.    300     IN      NS      ns0.ahfywbz.com.
windowsoxonline.com.    300     IN      NS      ns0.ckjdtybz.com.
windowsoxonline.com.    300     IN      NS      ns0.uthvfybz.com.
```

In botnet technology, this technique is used to connect and hide to the command and control servers (C&C) [13], like normal DNS (see figure 4). The big difference is that usually these DNS domains are stolen/hijacked, hacked or registered with (usually) fake creditcard data. It is also

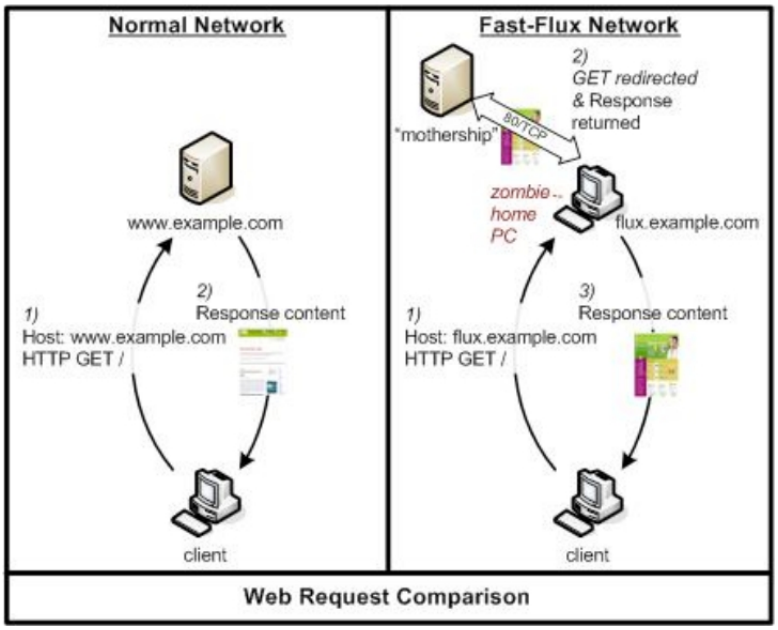

Figure 4: Fast flux networks (source: Honeynet Project [3])

used by botnets to hide phishing and malware sites behind a changing network of compromised hosts. These sites are used to deploy malware (botnet software like Peacomm) to unaware users. It can also refer to the combination of peer-to-peer networking, distributed command and control, web-based load-balancing and proxy redirection used to make malware networks more resistant to discovery and counter-measures [9].

How fast-flux networks work The fast-flux networks are a group of compromised (hacked) computer systems which have public DNS record[3]. These records change very fast, which makes the detection of those networks harder. Fast-flux networks have multiple (tens, hunderds, or] thousands) of IP addresses assigned to it [3]. These IP addresses in the A records are changing very fast, using round-robin IP adresses and a very short TTL [3]. An unaware user connecting to a website might be connecting to a different infected host each time. The IP address pool is usually not the final destination, they merely are redirectors that forward the requests to other backend servers, who serve the content. This technique is used for loadbalancing and high availability, but botnet herders have used this for illegitimate purposes.

The controlling elements are called “motherships” [3] by the researchers of the Honeynet Project. These motherships are hidden by the front-end fast-flux nodes. The motherships host both the DNS and HTTP services, and can be configured to manage thousands of domains simultaneously on a single host. The motherships provide the information for DNS to the frontend, who then forward it to the infected client. There are 2 different types of fast-flux networks: single-flux and double-flux [3] (see figure 5).

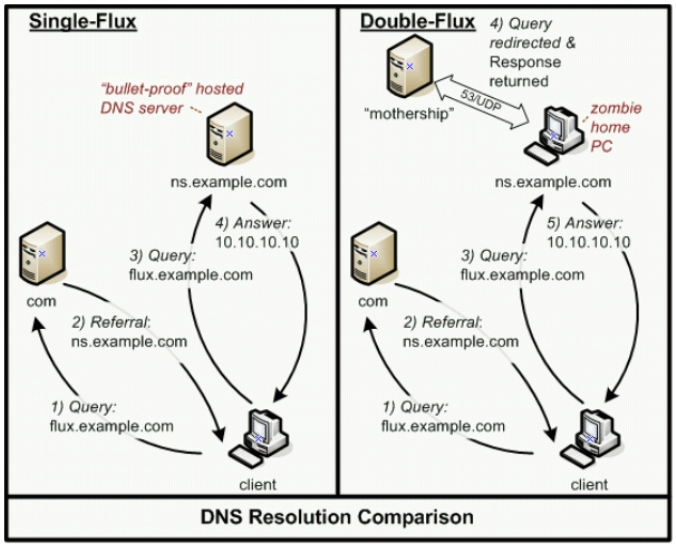

Figure 5: Single versus Double flux(source: Honeynet Project [3])

Single-flux Single-flux do not shield the “mothership” or “bullet-proof” hosted DNS server from the public. The client will make a request, and the DNS server will return a value. This value will change rapidly, but the DNS server will stay the same, making it more vulnerable for shutdowns [3].

Double-flux Double-flux are more complex providing another layer which does hide the mothership. The A record and the authorative NS record care continually changed and propegated. With double-flux the client makes a request, and gets a NS record. With single-flux this NS record was static, but now with double-flux, every request may get a different NS record. The DNS server runs on a zombie computer, hiding the mothership from view. When a client makes a request to the nameserver, the mothership provides the information to the zombie server, thus hiding the mothership, making it very hard to detect.

Howto detect fast-flux networks

It is very difficult to detect fast-flux network with your own network. The detection of fast-flux networks depends on analyzing the DNS requests made in your network. If an administrator sees a lot of requests for a unknown domain, it’s very probably used for fast flux. Also an administrator can look at the TTL, if it is extraordinary low, it is likely to be used for fast flux.

2.4.2 Peer-to-peer obfuscation

Botnet’s may make a lot of connections, with a lot of unsuccessful connections attempts. Sometimes this is a sideeffect, because botnets always change: bot’s leave, and bots join. On the other side, making bogus connections may be an attempt to try to obfuscate the location of the

command and control bots. By making a lot of connections, it takes more time to analyse the connections, thus taking it more time for administrators to track the C&C server. What reveals to an attempt to obfuscate is a sudden increase of connections. Trying to find a C&C server is a hard job, because you have to analyse each connection. Analysing on a greater scale, not per client, but for the whole network will ease this task. Finding out in the network where a lot of connections are made to, may point to a control and command server. This is however, time intensive.

2.4.3 Rootkits

A lot of bots use rootkits to hide their presence (installation files, registry entries and processes) [13] [23] by modifying the Windows API. A rootkit can also hinder anti-virus programs, especially if the engine and the database are old. Peacomm uses a rootkit that attempts to disable anti-virus programs, and denying processes started by the anti-virus program to start. Rootkits can be revealed clientside by special programs called "rootkit revealers". These programs compare the hard data on the harddisk, and the data which the operating system returns. Rootkits can also be discovered through the network by looking at unknown connections made from the client pc.

2.4.4 Low presence

Botnet herders, and particularly on Peacomm only use a small portion of the botnet to spread the infection. Because only a small part is spreading the infection, it will be hard for network administrators to notice the traffic in a large scale network.

2.4.5 Polymorphism

Polymorphism is very old in the world of viruses, but still anti-virus programs have problems detecting these methods. Polymorph malware self-mutates upon replication. New generation malware like Peacomm modify their binaries, and then the botnet herder will inject multiple versions into the botnet through one of the nodes. This makes it harder for anti-virus vendors to make fingerprints and even though anti-virus vendors have improved signature/heuristics delivery time down to several hours in some cases, the newest viruses take advantage of that unprotected time and point all ammunition to the vulnerability. The makers of Peacomm uses several strategies in polymorphing Peacomm: [25]

- Peacomm uses social engineering to get hosts infected and therefore modify the sources. For example: the days before valentines day they use "Valentines greetings", and at the end of the year "Merry christmas" and "Happy newyear".
- Low variant volume. Each variant is distributed in relatively small quantities or instances. Since an anti-virus vendor must be aware of a malware sample in order to analyze it in its laboratory, distribution in low numbers often enables the malware to fly below the radar of the traditional anti-virus engines.
- Brief variant lifetime. The fleeting lifetime of each variant is two to three hours on average, and each variant rarely makes a second appearance during the outbreak. Since it takes several hours to develop a new signature or heuristic, and up to several days to distribute to end users, the short-lived variants are typically out of distribution by the time traditional anti-virus defenses are available.
- Vast variant quantity. These malwares distribute a vast number of variants. Peacomm distributed more than 7,000 distinct variants on several days during New Year, and more than 40,000 altogether during a 12-day period. Since each variant or group of variants requires a different signature, it is impossible for anti-virus engines to keep up with this rapid-fire pace.

3 Case study: Peacomm

3.1 Introduction

Peacomm is the most recently known peer-to-peer bot to date and is the only one using a completely decentralized peer-to-peer protocol. The botnet uses the Overnet protocol (which is based on the Kademlia algorithm) for controlling the bots. The Kademlia algorithm uses a distributed hash table design, and because of this, there is no hierachy in the topology [26] [14] [31]. The kademlia algorithm is very resilient to lost peers, and simply patching over holes in the hash table, and thus cannot be shutdown easily.

The Kademlia algorithm is based on the calculation of the "distance" between two nodes. This distance is computed as the exclusive or of the two node ID's, taking the result as an integer number. Keys and Node ID's have the same format and length, so distance can be calculated among them in exactly the same way. This "distance" does not have anything to do with geographical conditions, but designates the distance within the ID range [11].

There are multiple network clients and networks that use the kademlia algorithm, but they are not compatible with each other [11].

1. Overnet network: eDonkey (no longer available) and MLDonkey (old),
2. Kad Network: eMule, MLDonkey (new clients) and aMule,
3. BitTorrent Mainline DHT: BitTorrent, uTorrent (v1.2 and higher), BitSpirit v3.0 (and higer) and many libtorrent-based: They all share a DHT based on an implementation of the Kademlia algorithm, for trackerless torrents.

In 2006, the Overnet and eDonkey2000 were shutdown by the RIAA. The RIAA is the Recording Industry Association of America, a tradegroup in the United States, who protect the legal rights of musicians and entertainers. But the Peacomm botnet continued to use the Overnet protocol.

Hash tables are a datastructure that associate keys with values [10]. The primary operation it supports efficiently is a lookup: given a key (e.g. a node), find the corresponding value (e.g. the node's IP address). Peacomm uses distributed hash tables to find other nodes. The whole network is sliced up in parts, and each node knows about a part of the network, hence the name distributed. After *infection*, Peacomm uses a hard-coded list to find the initial nodes to the botnet. As it is making succesful connections, it receives more information about the botnet from other nodes. With each succesful connection, it will save the hash (node information) into local memory first. Peacomm then searches for the second injection using a hash to get into the second phase.

The secondary injections are downloaded from the peer-to-peer network, which provides a basic communication primitive from the attacker to the infected machines. This primitive enables the attacker to upgrade, control, or command infected hosts without relaying on a *central* server [14].

Below is our Peacomm system and network analysis.

3.2 Test environment

We first analyzed the bots infection mechanism in our test environment before starting monitoring its behavior. Our test environment is setup in a secure environment to avoid infecting other machines or participating in malicious attacks. We used a Peacomm.D specimen (`dancer.exe` with detection date January 2008) for our case study. This specimen is an newer variant of Peacomm.D and at the time or writing little is known about this variant. The initial executable was provided by Nicolas Albright (`http://www.disog.org`).

Our test setup consists of one software firewall installed with IPCop [22]. Three computers, installed with Windows XP SP2, are all connected through a hub to the firewall. One host will be logging all the traffic with Wireshark [6]. The other two host will serve as zombies in our botnet (see figure 6).

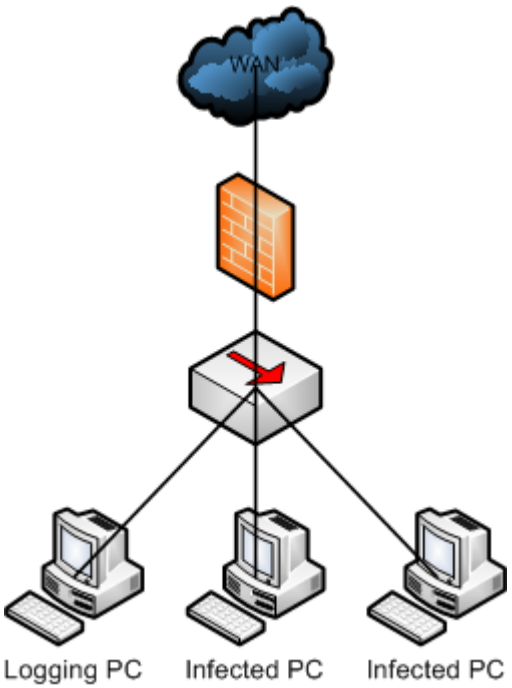

Figure 6: Our test environment

The infected hosts are all installed with PerilEyez [16] and SysAnalyzer [44]. These tools are used to detect changes in the system. We also installed RootkitRevealer[34], Rootkit Unhooker [17] and Windows Malicious Software Removal Tool [28] to see if these programs could detect the malicious software installed on the hosts.

3.3 Social engineering

The peacomm trojan comes in a variety of flavors. All these methods are used to confuse users and anti-virus vendors. Users are then enticed to click on attachments or files coming through websites or email. By clicking, they download the trojan and become infected.

Iframed[4] in or attached to mails. The most common method, the peacomm bot sends out a HTML spammail, and hides a link in a hidden iframe that will download the peacomm bot.
Other methods are displaying the YouTube logo, and hiding a file called video.exe [5] behind it.

"Main event" outbreaks: the peacomm is modified to a coming main event, like the NFL sports league opening in the United States of America. A fake NFL sports (figure 7 on the following page) tracker is setup, and the peacomm trojan is hidden in the tracker[6]. The trojan is named

[4] http://www.disog.org/2007/09/stormworm-iframe-hell.html
[5] http://www.disog.org/2007/08/dude-what-if-your-wife-finds-this.html
[6] http://www.disog.org/2007/09/storm-domains-locally-resolving.html

`NFLTracker.exe` and is identified as "Trojan.Peed.III". When unknowning users click on the tracker, it is activated. To lure the user to the site, spammails are generated:

```
Subject: Get Your Free NFL Game Tracker

Football......Need we say more?
We want you to have all the details for every game this football season.
Get all the info you need from our online game tracker:
http:// 58 . 79 . 110 . 12
```

Figure 7: A fake NFL tracker

Fixed date outbreak: dates that are already known and popular like labor day [7], valentine's day[8], chrismas [9] and new years day[10] are also targeted.
The notice comes via mail, are users are lured to the legit looking domains, for example newyearcards2008.com and happycards2008.com (as of writing offline). These fake sites (figure 8) then serve files with innocent names like `happynewyear2008.exe` or `happy_2008.exe`.

Figure 8: A peacomm deployed Merry Chrismas site

Exploiting security holes: the Peacomm creators exploit security holes in popular applications. For example, they are known to exploit Realplayer and Quicktime to point to an iframe, containing encoded javascript which then installs the Peacomm bot [11].

Peacomm lifting with popular file formats: Peacomm is also known to infect certain .mp3 files and email it out to users [12].

[7] `http://www.disog.org/2007/09/latest-stormworm-filenames.html`
[8] `http://www.disog.org/2008/01/cme711-happy-valentines-day-and-halifax.html`
[9] `http://www.disog.org/2007/12/stormworm-is-back-have-merry-christmas.html`
[10] `http://www.disog.org/2007/12/new-year-recycled-greeting-cards.html`
[11] `http://www.disog.org/2007/12/quicktime-and-realplayer-exploits.html`
[12] `http://www.disog.org/2007/10/mp3-pump-and-dumps.html`

Peacomm is pretending to be another program: Peacomm's creators setup a website with the pretence of announcing a new P2P filesharing program. The page advertises a P2P application called Krakin, which, among other things is said to be: Easy to install, prevents tracking, has blogs and chat platforms, and video mail. Of course, when an user downloads it, he will be infected.

3.4 Infection

We first used CWSandbox [36] to analyze our peacomm.D specimen. A detailed report is included in attachment A.2 on page 37 but this doesn't include all the peer lines in the INI file `C:\WINDOWS\noskrnl.config` . The complete list can be obtained at request. We also used our tools (PerilEyez [16] and SysAnalyzer [44]) to analyze system changes.

The Peacomm.D binary is an executable that installs the initial bot on our host. The binary is distributed as a Trojan horse email attachment, in our case `dancer.exe`. It can be installed on different Windows platforms namely Windows 95/98/ME/2000/NT/XP [5]. When executing, the file `dancer.exe` is copied to `c:\windows\noskrnl.exe` (after getting the Windows directory) and creates the following registry entry so it runs when Windows starts:

```
HKEY_CURRENT_USER\Microsoft\Windows\CurrentVersion\Run\"noskrnl" = "%Windir%\noskrnl.exe"
```

After looking for the files `w32tm.exe` and `noskrnl.exe`, 3 files are executed:

- the time configuration:

  ```
  w32tm /config /syncfromflags:manual /manualpeerlist:time.windows.com,
  time.nist.gov
  ```

- the time synchronization execution: `w32tm /config /update`
- the execution of peacomm: `noskrnl.exe`

During the `noskrnl.exe` process, we could distinguish following important actions:

- File system changes:

  ```
  Find File: netsh.exe
  Create File: C:\WINDOWS\noskrnl.config
  ```

- The INI file `C:\WINDOWS\noskrnl.config` will be read. The hard-coded list with peers is written in this file and each line is a single hex-encoded peer in this format:

  ```
  <128 bit hash>=<32 bit ip><16 bit port><8 bit peer type>
  ```

  An extract of our list, would look like this:

  ```
  Write to INI file: C:\WINDOWS\noskrnl.config [peers] 8E037D228152C4616E6100089D6AE30B
   = 84EF0172CC9300
  Write to INI file: C:\WINDOWS\noskrnl.config [peers] 8F039554595E5E6531235D3746467E53
   = 4A4F6716643E00
  Write to INI file: C:\WINDOWS\noskrnl.config [peers] 9003D16E06508441F7572B42960B6513
   = 4118D89B0F9200
  Write to INI file: C:\WINDOWS\noskrnl.config [peers] 91038E7FE449442EDF0B963BA909882A
   = 3E9496B30B9900
  Write to INI file: C:\WINDOWS\noskrnl.config [peers] 92039F352023206D1D5E6A6E5B2C013C
   = A80B2002272400
  Write to INI file: C:\WINDOWS\noskrnl.config [peers] 93035346BC044A02477A342FAE67BA16
   = 522DDB4D270F00
  Write to INI file: C:\WINDOWS\noskrnl.config [peers] 94032332D35C2D20F85EFE66DD4A5222
   = 84EF0172CC9300
  ```

In our specimen, the full list contained a 1000 lines initial peer list. When bootstrapping, Peacomm doesn't contact all the peer nodes but uses a random list of peers. If it doesn't get an answer from any of the peers, Peacomm will sleep for 10 minutes and restarts the process of contacting it peers [31]. At the bottom of the list, we also see that the UDP port is written to the `noskrnl.config` file:

```
Write to INI file: C:\WINDOWS\noskrnl.config [local] uport = 17097
```

It must be noted that this port is randomly chosen on every infected host. Once it is written to the `noskrnl.config` file this port will always be used by the malware.

- In earlier version Peacomm disabled the firewall. To be less suspicious, it now creates a rule in the Windows Firewall to allow `noskrnl.exe` to access the network:

  ```
  netsh firewall set allowedprogram C:\WINDOWS\noskrnl.exe enable
  ```

- To hide its process, it also loads a driver `System32\noskrnl.sys` in `services.exe`:

  ```
  (\Registry\Machine\System\CurrentControlSet\Services\noskrnl.sys) File Name: ()
  in services.exe
  ```

- The service `Windows\noskrnl.exe` acts as a peer-to-peer client and downloads the secondary payload injections [5].

The first packages sent by Peacomm are for the bootstrap process to become part of the Overnet network [14]. This can be done by the earlier mentioned hard-coded list in the INI file `C:\WINDOWS\noskrnl.config`. This list is frequently being updated with connected peers. We could also see that the port always stays the same even after a reboot.

3.5 Peacomm communications

Peers in the overnet network use other peers to route searches and data [8]. The search is used to locate the secondary injection (Phase 2), or to update itself. The search hashes Peacomm uses to locate the secondary injection is not static, thus cannot be easily fingerprinted. They are generated based on a random integer between 0 and 31, and the current system time. This is why we saw the NTP connection being made during phase 1 in the experiment.

The second injection consists of multiple stages, that can differ per client [38]. One client can be configured to participate in a distributed denial of service (DDOS), while the other can recieve a command to send out spam.

The eDonkey 2000/Overnet protocol uses a UDP packet to announce itself to the network, and TCP to exchange data between peers. The eDonkey/Overnet servers exchange serverlists with: 0xa4 (Request) and Reply (0xa1). It also pings (0x96) and announces itself with message type 0xa0. The reply to the pings is 0x97 (also called pong) and contain the users and files count as well. There are also more message types (0xxx) to exchange data, queries and search results described by Oliver Heckmann and Axel Bock [30].

The Peacomm botnet uses eleven messages (See table 3 on the next page). Because it is encrypted, and packet analyzers like Wireshark can't look into the packets, we assume it has some similarities with the original Overnet protocol. Peacomm only uses UDP communication with other peers. This makes it easier to detect: the unusual amount of UDP packets in a very short time span can be very clearly seen with flow analysis. We found several unique packets during phase 1 using Wireshark[6].

```
Attempting to connect to other peers:
Edonkey protocol: Unknown (010)
Message type: Unknown (0xa6)
```

And during phase 2 (among others):

```
Attempting to connect to other peers (61 bytes)
Edonkey protocol: Unknown (010)
Message type: Unknown (0xa6)

Attempting to connect to other peers (67 bytes)
Edonkey protocol: Unknown (010)
Message type: Unknown (0xa6)

Server list (different, found: 345,414, 506 bytes)
Protocol: Unknown (010)
Message type: Server list (0xa1)
```

We investigated the message types, and identified 11 different message types. (See table 3).

Packet type	Occurance	Traffic	Size	Percentage
0xa0	1224	Mainly outgoing	67 bytes	3.7556%
0xa1	1168	Mainly incoming	range 200-400 bytes	3.5838%
0xa4	140	Mainly outgoing	61 bytes	0.4296%
0xa5	54	Mainly incoming	range 100-268 bytes	0.1657%
0xa6	22280	Mainly outgoing	67 bytes	68.3624%
0xa7	7526	Mainly incoming	60 bytes	23.0922%
0xb1	8	Outgoing	60 bytes	0.0245%
0xb6	5	Incoming	60 bytes	0.0153%
0xb8	50	Incoming	60 bytes	0.1534%
0xba	51	Outgoing	65 bytes	0.1565%
0xbb	85	Incoming	80 bytes	0.2608%

Table 3: Wireshark packet count

With the Wireshark filter `edonkey.message.type == 0xa1` you can filter the message types. When looking at the packets, the type 0xb6 is very rare as you can see in table 3. There is a total of 32591 eDonkey packets in phase 2, 99.9998% is accounted for, the rest does not contain an eDonkey message type and can not be identified correctly. Because the packets are encrypted and the communication of Peacomm is not open, we cannot say which packet type contains which data.

3.6 Comparison with legimate P2P traffic

To see if the traffic which is coming from a node, is something other then Peacomm's communication, for example legit peer-to-peer traffic. We made a comparison with other peer-to-peer traffic which are the most popular and might resemble peacomm's communication the most: eMule, Bittorrent and Direct Connect. With all the P2P programs, we connected to the network: eMule [13] to the eDonkey/KAD network, uTorrent[14] to a Ubuntu tracker and DC++ to random hubs. On eMule and DC++[15] we searched for Ubuntu ISO's and downloaded them partially to see the traffic. On Bittorrent we downloaded a .torrent file from a Ubuntu mirror.

[13] `http://www.emule-project.net/`
[14] `http://www.utorrent.com/`
[15] `http://dcplusplus.sourceforge.net/`

3.6.1 Bittorrent traffic

We used uTorrent version 1.6.1, and configured it to work on ports TCP port 30001. We kept the bittorrent connection open for 66.166 seconds, and captured 31345 packets. The result was 0.30% UDP traffic (95 packets), the majority were TCP packets: 31215 packets (99.59%). The rest was ICMP, DEC DNA routing protocol and Microsoft Network Load balancing traffic. Of the TCP connections, 3359 packets (10.76%) of the total TCP packets were Bittorrent, the rest being point to point traffic. The rest of the packets have nothing to do with the P2P connection like STP, ARP and regular NETBIOS traffic. See also table 4.

Conclusion: The Bittorrent traffic is very different from Peacomm traffic. There is virtually no UDP traffic involved with Bittorrent. The only similarity is that Bittorrent also creates a lot of packets in a very short timespan.

Packet type	Occurance	Percentage
Total	31345	100%
UDP	95	0.30%
TCP DATA	31215	99.59%
Bittorrent protocol	3359	10.72%

Table 4: Wireshark Bittorrent packet count

3.6.2 eMule traffic

eMule uses the same type of communication as Peacomm/Overnet, they are both based on the Kademlia algorithm and the eDonkey protocol.

We used client 0.47, and configured eMule to work on TCP port 30001 and UDP port 30002. We connected to the eMule/eDonkey network on random hubs and searched and downloaded random Ubuntu ISO's. We kept the connection for 167 seconds and captured 3997 packets.
The result was 10.23% UDP traffic (409 packets), of which 33.25% (136 packets) were eMule protocol packets. The peer-to-peer communication was almost entirely TCP: 89.77% (3588 packets). The rest of the packets have nothing to do with the P2P connection like STP, ARP and regular NETBIOS traffic. See also table 5.

The eMule/eDonkey traffic differs very much from Peacomm communication, which is almost entirely UDP except for some TCP packets (which are rare).

Packet type	Occurance	Percentage
Total	3997	100%
UDP	409	10.23%
eMule	136	3.40%
TCP	3588	88.77%

Table 5: Wireshark eMule packet count

3.6.3 DC++ traffic

We used DC++ version 0.691, and configured it using TCP port 30001 and UDP port 30002. We kept the DC++ connection open for 72.123 seconds and captured 1222 packets. Of these packets, 23.98% (293) packets were UDP. The majority is TCP with 73.73% (901 packets). The rest of the packets have nothing to do with the P2P connection like STP, ARP and regular NETBIOS traffic. See also table 6 on the next page.

Packet type	Occurance	Percentage
Total	1222	100%
UDP	293	23.97%
TCP	901	73.73%

Table 6: Wireshark DC++ (Direct Connect packet count)

The DC++ uses more UDP then the other protocols, and also uses plaintext to pass messages and the HUB lists to the DC++ client. The UDP traffic mainly consists of search packets.

```
$SR [Other]shore SHARED2\ubuntu-6.06.1-server-i386.iso.torrent34805
12/12TTH:XAWDBHU5LH2K4TEKWQDC7FDEZT6ELOL2BQAOBII (89.248.168.91:6969)|
```

Although there are not a lot of TCP data packets (the being plaintext packets), there are a lot bigger (ranging from a hundred bytes to thousands of bytes).

The telnet messages pass messages, hublists and clientlists to the client. Passing the Hublists to the user:

```
Lock EXTENDEDPROTOCOL_verlihub Pk=version0.9.8c|<-B1-Security-> This Hub is running
version 0.9.8c BatMax. Network (Mon Jan 22 16:00 GMT 2007) of VerliHub (RunTime:3day
s 4hours ).|$Supports UserCommand NoGetINFO NoHello UserIP2 TTHSearch ZPipe0 GetZBlo
ck |$Key u.......A .....01.1q.P..q|$ValidateNick Meri|<-B1-Security-> This hub is en
hanced by plugman for Verlihub.|$Supports OpPlus NoGetINFO NoHello UserIP2|$HubName
[..BatMax][1..][15GB]     -  [..BatMax][1..] [ Type   ./fav    .In Main To Add This
Hub To Favo Hubs ]|$Hello Meri|$Version 1,0091|$GetNickList|$MyINFO $ALL Meri 6950 G
B<++ V:0.691,M:A,H:1/0/0,S:5>$ $0.005.$blaat@aap.nl$538461095766$|$MyINFO $ALL [LV]s
amuraii $ $0.005.$$17245842655$|$MyINFO $ALL Julenissen $ $0.01.$$99393763785$|$MyIN
FO $ALL [Telia]Azrael $ $DSL.$$722961864418$|$MyINFO $ALL bingobongo $ $0.5.$$121242
679504$|$MyINFO $ALL novafire $ $DSL.$$31467653240$|$MyINFO $ALL Nocturath $ $0.5.$$
106096475404$|$MyINFO $ALL (BBB)sirken $ $1.$$56600354956$|$MyINFO $ALL (adsl)legio
$ $5.$$0$|$MyINFO $ALL [10MBIT]ReginaDawn $ $10.$$10622056047$|$MyINFO $ALL ifialka
$ $2.$$31232605724$|$MyINFO $ALL musicalkey76 $ $10.$$334379125001$|$MyINFO $ALL The
cat $ $1.$$139573315867$|$MyINFO $ALL NL[8/1]Kiwi $ $Cable.$$55636157230$|$MyINFO $A
LL backspace64 $ $0.005.$$63282062213$|$MyINFO $ALL Daftman $ $LAN(T1).$$16965534660
8$|$MyINFO $ALL visavis1981 $ $0.005.$$124862533982$|$MyINFO $ALL [CH]poliallez $ $0
.2.$$180665677151$|$MyINFO $ALL Bigpat $ $0.005.$$120933343257$|$MyINFO $ALL LaZa[SK
] $ $100.$$70755672238$|$MyINFO $ALL [HUN]don101 $ $2.$$81536300842$|$MyINFO $ALL Ea
mon $ $Cable.$$48563180075$|$MyINFO $ALL Mantas $ $0.005.$$23207014400$|$MyINFO $ALL
 mafka $ $5.$$57219559750$|$MyINFO $ALL Airship22 $ $0.5.$$69348085765$|$MyINFO $ALL
 P4LL1Magotti $ $Cable.$$3022119269$|$MyINFO $ALL [FIN]Jussi76 $ $DSL.$$38322487419$
|$MyINFO $ALL [Telia]Rhex $ $DSL.$$12133173240$|$MyINFO $ALL Koklis $ $0.005.$$30302
142713$|$MyINFO $ALL Herm $ $DSL.$$866119323
```

Passing the messages to the client:

```
$Lock EXTENDEDPROTOCOL_verlihub Pk=version0.9.8c|
<-B1-Security-> This Hub is running version 0.9.8c BatMax
Network (Mon Jan 22 16:00 GMT 2007) of VerliHub (RunTime:3days 4hours ).|
```

Plaintext connections are also used to initiate a connection to another client:

```
$ConnectToMe [Sunet]Lolers 82.75.71.244:30001|
```

Conclusion: Direct Connect traffic is also very different from Peacomm's traffic, with almost 75% TCP traffic. Also the usage of (lots of) plaintext packets are unique to Direct Connect, Peacomm does not use plaintext. Also Direct Connect still needs a central source called a hub to connect to other users, opposite of Peacomm which is fully decentralized. Direct Connect users also initiate connections with plaintext packets, which is not encrypted and human readable. These connection initiation packets are variable, Peacomm uses the same strings to connect to other nodes.

3.6.4 Online game traffic

We have conducted this experiment with an online game: Day of Defeat[16]. This game uses Steam technology, where a lot of other very popular games like Half-life, Counter-Strike and Team Fortress are built upon.
We have monitored the connection for 150.19 seconds, and captured 7092 packets. The result was 57.45% UDP traffic (4060 packets) and 36.97% TCP traffic (2613 packets). The rest was ICMP, common network traffic like Spanning Tree and ARP, and Microsoft NETBIOS traffic.
Although the user is playing with a lot of other human and computer players from all over the internet, all the connections go through the Day of Defeat game server. The packets are all UDP, and the source and destination ports are static. The destination port of the game server is configured by the administrator and thus static. The source port of the user is always between 27000 and 27030, this is because of the game protocol.
When looking into the traffic, the packet sizes are variable, ranging form 60 to 300 bytes.

Conclusion: Online gaming is very much different from Peacomm, the only similarity is that it also uses UDP and static ports for communications. The main differences are that the online game's packetsizes are variable, and the traffic only originates from 1 source: the gameserver.

Packet type	Occurance	Percentage
Total	7067	100%
UDP	4096	57.45%
TCP	2613	36.97%
ICMP	277	3.92%
IGMP	10	0.14%
Other	107	1.52%

Table 7: Wireshark online game packet count

3.7 Secondary injections

In his infections stage, Peacomm isn't designed to act as malicious software. As we already mentioned before, the malware will first contact his peers. After establishing a connection with the peers using the Overnet protocol, the second payload injection can begin. In the second stage Peacomm will assign a task carried out by the bot. Primarily, following actions *can* be undertaken by the bot [14]:

- download latest version of the malware
- spamming component
- distributed denial of service tool

At the second stage we can see, using PerilEyez [16], that the malware (in our tested sample) will duplicate himself in every *sub*directory on the desktop:

```
+ File C:/Documents and Settings/p2pbotnets/Bureaublad/malware_tools/
_install.exe was added
+ File C:/Documents and Settings/p2pbotnets/Bureaublad/malware_tools/XXX/releases/
_install.exe was added
+ File C:/Documents and Settings/p2pbotnets/Bureaublad/malware_tools/XXX/Modules/
_install.exe was added
```

[16] http://www.dayofdefeatmod.com/

```
+ File C:/Documents and Settings/p2pbotnets/Bureaublad/malware_tools/XXX/releases/
ScreenShot/_install.exe was added
+ File C:/Documents and Settings/p2pbotnets/Bureaublad/malware_tools/XXX/Modules/
_install.exe was added
```

At this stage, we could also detect a lot of traffic from a high numbered UDP and TCP (for sending spam) port. A more in depth analysis about the network traffic can be found in section 3.8. During this network analysis, we saw that our bot was assigned to engage in sending spam. With Winhex [2] we could access the physical RAM and distinguish the message send to the email recipients:

```
ived: (qmail 12996 invoked from network); Thu, 24 Jan 2008 15:48:52 +0100
Received: from unknown (HELO cgyuq) (58.45.119.99)
by exp16.studexp.os3.nl with SMTP; Thu, 24 Jan 2008 15:48:52 +0100
Message-ID: <002601c85e98$3c6a79b0$63772d3a@cgyuq>
From: <mcriscitiello@goingplaces.co.ke>
To: <pr@litsmart.com>
Subject: %^Fpharma^%
Date: %^D-%^R30-600^%^%
MIME-Version: 1.0
Content-Type: text/plain;
format=flowed;
charset="%^Fcharset^%";
reply-type=original
Content-Transfer-Encoding: 7bit
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 5.50.%^C7%^Foutver.5^%^%
X-MimeOLE: Produced By Microsoft MimeOLE V5.50.%^V7^%

%^G%^Fpharma^% http://%^P%^R2-7^%:qwertyuiopasdfghjklzxcvbnmeuioaeuioa^%.
%^Fpharma_links^%^%
```

The same message was captured with Wireshark [6]. Because we weren't able to find the message on disk, we assume the malware only loads the message in RAM. When we were looking for this, we also found an update for the peer list. This was also loaded in RAM but not updated to the `noskrnl.config` file. After a reboot, we could see the `noskrnl.config` list was updated.

3.8 Network analysis

We examined the traffic generated on a production machine infected with Peacomm. This traffic represents different sessions, during the 2 weeks we have tested Peacomm, and contains packets from our 2 infected bots, our logging pc and our firewall / DHCP server / router.

3.8.1 Infection

We blocked all the traffic from our infected host to see what traffic it will initially generate and to identify Peacomm specific traffic. In a time frame of 3 minutes, we could distinguish a lot of UDP traffic. The host starts with contacting `time.windows.com`. This is not explicit malware traffic but is a consequence of configuring the time protocol in the infection stage (see section 3.4 on page 17). More then 85 % is Peacomm traffic trying to contact his initial peers (see table 8 on the following page). Per host, the UDP discovery packets, send by Peacomm, all use the hard-coded port number (for this host 27978), as configured in their `noskrnl.config` file, and the same packet size (64 bytes). If it can't contact his peers, it will stop sending UDP packets.

UDP	% Packets	Packets	Bytes
Bootstrap Protocol	1,18%	2	698
Domain Name Service	2,96%	5	380
Peacomm traffic	86,98%	147	9849
NetBIOS Datagram Service	1,78%	3	648
NetBIOS Name Service	7,10%	12	1104

Table 8: UDP hierarchy statistics (generated with wireshark [6])

In figure 9, we have an overview of the total initial traffic. We did this by using a torrent client (uTorrent [41]), browsing the Internet and performing office tasks. We can conclude that most of the traffic is UDP traffic. The first peak is caused by Peacomm's peer discovery. In this case, he couldn't find a connection whereafter all the UDP traffic dies. Peacomm will try this every 10 minutes. Connection with one peer is enough to update his peer list and engage in malicious activities.

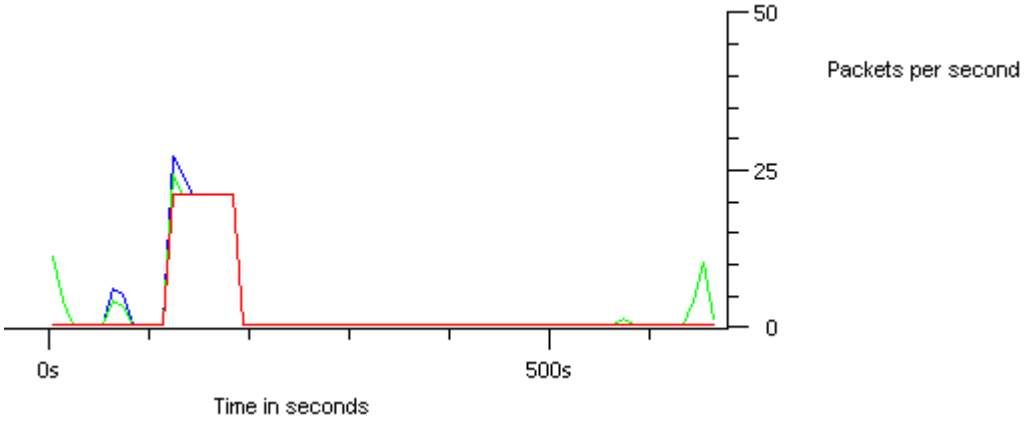

Figure 9: Relationship between all traffic (blue), UDP traffic (green) and *Peacomm* traffic (red) (generated with Wireshark [6])

3.8.2 Production environment

After Peacomm's infection we tried to create a network trace of a regular production host. Our goal was to look if traffic generated by the malware could be easily detected. In section 3.8.1 on the preceding page we could already distinguish some unique characterizations in the network flow. Now, we will take it one step further.

If we take a look at table 10 on page 26, we see some interesting numbers:

UDP traffic We see that the UDP traffic (in packets) is almost 55 %. This is due to the noisy traffic generated by the malware. In bytes, overall UDP traffic is only 10 % of the total amount of bytes transferred. This is due to the torrent traffic generated. With the help of the previous section, we know now that Peacomm will send a lot of UDP packets and will always use the same host port. Knowing this, we generated the table using this knowledge and see that this traffic is a little more over 52 %. Production hosts with a lot of UDP traffic are rare because the most traffic nowadays are encapsulated with TCP [35].

After analyzing packet sizes in malware generated UDP traffic, we were very much surprised to see the results as showed in table 9 on the following page. Approximately 98,5 % had a

Packet Length	Count	Percent
0-19	0	0%
20-39	0	0%
40-79	154681	98,53%
80-159	194	0,12%
160-319	1266	0,81%
320-639	841	0,54%
TOTAL	156982	100%

Table 9: Packet sizes in malware generated UDP traffic (generated with wireshark [6])

packet length between 40 and 79. This means that *Peacomm specific* traffic (in size and in port number) would be 51,6 % of all traffic in our network trace. The ports Peacomm uses are not officially assigned ports [21].

ICMP ICMP packets are also inherent to the Peacomm protocol, although maybe less obvious. Only 0,72 % is ICMP traffic and can be easily overlooked because this traffic isn't inherent to malware. In normal circumstances, a host shouldn't get so many ICMP requests. Not all peers send these requests. The cause why some of these malware peers send ICMP packets is unclear.

SMTP We can see that SMTP traffic kicks in at about 400 seconds. This is caused by Peacomm. He will first try to connect to his peers before getting an assignment. How this process exactly works is unclear because traffic is encrypted. If we take the total amount of packets

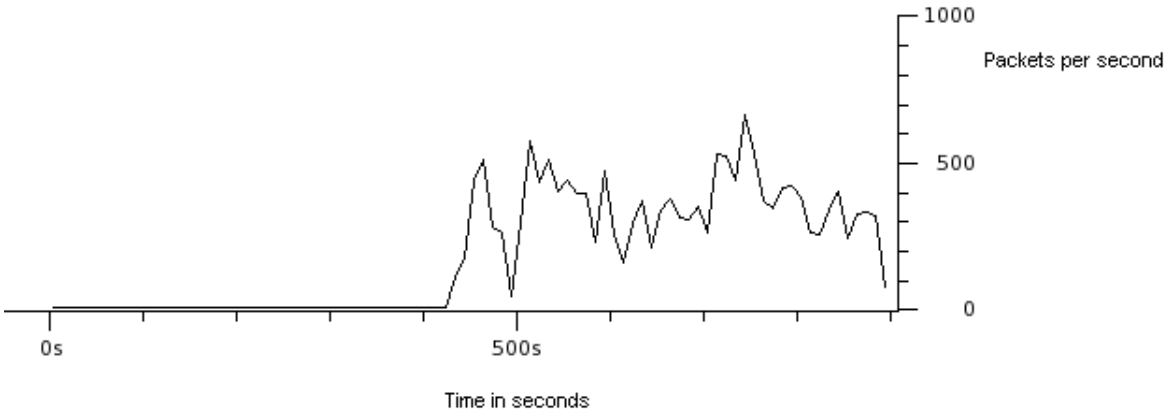

Figure 10: SMTP packets generated during the network analysis (generated with Wireshark [6])

sent through SMTP and take the time frame into account, SMTP sends 32,42 packets per second. Concluding that almost 5,5 % of the total traffic is SMTP, this leads to suspicion as these numbers are a lot higher then with regular hosts [35].
Another interesting fact is that Peacomm made 535 connections to unique remote SMTP servers when sending spam during a time period of 500 seconds. In time, the amount will decrease when the number of uncontacted smtp servers saturates. Peacomm will still make a lot of connections, see 3.8.3 on page 27.

DNS There are a lot of MX queries but this was less then 1 % of the total traffic. These packages are a constant value as we can see in figure 11 on the following page. As with the SMTP, MX queries kick in around 400 seconds. As from here, these queries are constant and are

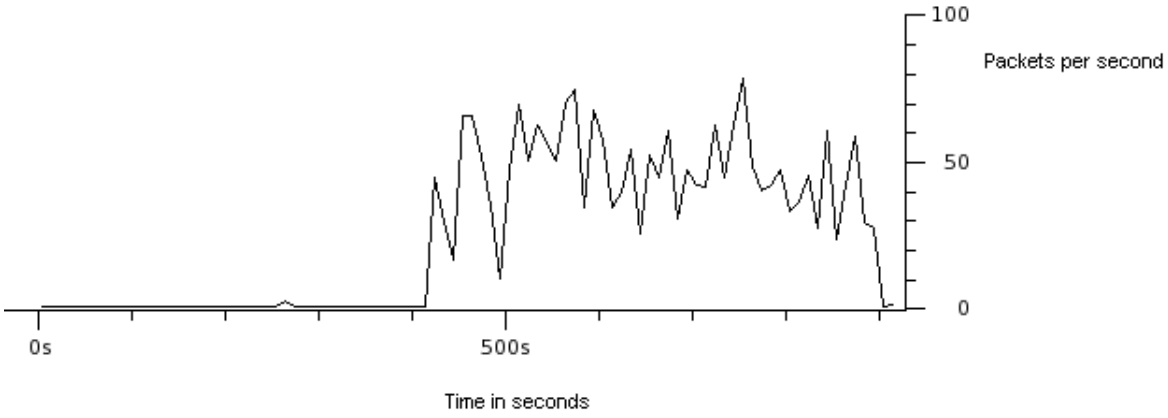

Figure 11: MX queries generated during the network analysis (generated with Wireshark [6])

suspicious. To rule out single isolated cases, we calculated the MX queries done over a period of 500 seconds. We got about 2225 packets between 400 seconds and 900 seconds. This means 4,45 packets per second including query and response. Looking at these statistics, we can't talk about a single case anymore. Production hosts doing such a high number of MX queries are definitely suspicious. MX queries aren't normally handled by these hosts.

Protocol	% Packets	Packets	Bytes	Mbit/s
UDP	*53,79%*	*161249*	*11180367*	*0,097*
Bootstrap Protocol	0,00%	1	342	0,000
NetBIOS Name Service	0,02%	60	5790	0,000
NetBIOS Datagram Service	0,00%	11	2511	0,000
Domain Name Service	0,99%	2955	403007	0,004
Network Time Protocol	0,00%	2	180	0,000
Malware traffic	*52,36%*	*156982*	*10532935*	*0,092*
Data	0,41%	1229	234180	0,002
Hypertext Transfer Protocol	0,00%	4	568	0,000
Quake III Network Protocol	0,10%	304	19613	0,000
Web Cache Coordination Protocol	0,00%	1	109	0,000
Network Service over IP	0,00%	2	455	0,000
Microsoft Media Server	0,00%	1	145	0,000
Distributed Interactive Simulation	0,00%	1	145	0,000
IGMP	*0,00%*	*8*	*480*	*0,000*
ICMP	*0,72%*	*2167*	*189332*	*0,002*
TCP	*45,49%*	*136367*	*96683436*	*0,841*
Hypertext Transfer Protocol	0,23%	681	445355	0,004
Secure Socket Layer	0,12%	363	205559	0,002
Data	1,07%	3202	4175378	0,036
BitTorrent	18,43%	55244	80699997	0,702
Simple mail transfer Protocol	5,41%	16212	1987531	0,017
TOTAL	*100,00%*	*299791*	*108053615*	*0,940*

Table 10: Protocol hierarchy statistics (generated with wireshark [6])

3.8.3 Open connections

The following list is an extract and is used to update the bot. Peacomm makes a connection and then the secondary payloads are downloaded through one of these hosts, in this case `host-n3-81-80.telpol.net.pl:2043`.

```
TCP    p2pbotnet-host2:1055   host-n3-81-80.telpol.net.pl:2043  ESTABLISHED
TCP    p2pbotnet-host2:1056   89.0.128.90.dynamic.barak-online.net:8811  TIME_WAIT
TCP    p2pbotnet-host2:1057   89.25.99.8:27948       TIME_WAIT
TCP    p2pbotnet-host2:1061   chello084010025030.chello.pl:7477  TIME_WAIT
TCP    p2pbotnet-host2:1062   80.232.251.157:20957   TIME_WAIT
TCP    p2pbotnet-host2:1063   c9537e3b.virtua.com.br:10788  TIME_WAIT
TCP    p2pbotnet-host2:1067   190.77.181.192:26417   TIME_WAIT
TCP    p2pbotnet-host2:1068   200.93.110-198.dyn.dsl.cantv.net:18894  TIME_WAIT
TCP    p2pbotnet-host2:1069   201-211-137-181.genericrev.cantv.net:17963  TIME_WAIT
TCP    p2pbotnet-host2:1057   89.25.99.8:27948       TIME_WAIT
TCP    p2pbotnet-host2:1061   chello084010025030.chello.pl:7477  TIME_WAIT
TCP    p2pbotnet-host2:1062   80.232.251.157:20957   TIME_WAIT
TCP    p2pbotnet-host2:1063   c9537e3b.virtua.com.br:10788  TIME_WAIT
```

The following list is an extract of connections with open mail relays. The goal is to send spam through these connections. Peacomm can actually use a great range of ports starting of at port 1025. We think this is because some ISP's block all incomming traffic on ports lower then 1024. We have seen that this range stops at port 1299.

```
TCP    p2pbotnet-host2:1078   85.110.245.105:6748    TIME_WAIT
TCP    p2pbotnet-host2:1079   88.200.145.226:10462   TIME_WAIT
TCP    p2pbotnet-host2:1080   gsmtp183.google.com:smtp  TIME_WAIT
TCP    p2pbotnet-host2:1081   bd224e67.virtua.com.br:20339  TIME_WAIT
TCP    p2pbotnet-host2:1095   mx3.hotmail.com:smtp   TIME_WAIT
TCP    p2pbotnet-host2:1105   dshs-spa.dshs-koeln.de:smtp  TIME_WAIT
TCP    p2pbotnet-host2:1106   207.159.120.164:smtp   TIME_WAIT
TCP    p2pbotnet-host2:1121   mail.global.frontbridge.com:smtp  TIME_WAIT
TCP    p2pbotnet-host2:1140   mta-v8.mail.vip.mud.yahoo.com:smtp  ESTABLISHED
TCP    p2pbotnet-host2:1218   scc-mailrelay.att.net:smtp  ESTABLISHED
TCP    p2pbotnet-host2:1225   mailhub-new.vianetworks.nl:smtp  TIME_WAIT
TCP    p2pbotnet-host2:1226   mxs.mail.ru:smtp       SYN_SENT
TCP    p2pbotnet-host2:1230   mxs.mail.ru:smtp       ESTABLISHED
TCP    p2pbotnet-host2:1238   mx3.hotmail.com:smtp   ESTABLISHED
TCP    p2pbotnet-host2:1240   mta-v8.mail.vip.mud.yahoo.com:smtp  ESTABLISHED
TCP    p2pbotnet-host2:1241   box48.bluehost.com:smtp  TIME_WAIT
TCP    p2pbotnet-host2:1242   gw.zedo.com:smtp       SYN_SENT
TCP    p2pbotnet-host2:1243   mx1.empal.com:smtp     ESTABLISHED
TCP    p2pbotnet-host2:1244   mail.laderma.com.au:smtp  ESTABLISHED
TCP    p2pbotnet-host2:1245   66.255.200.7:smtp      TIME_WAIT
TCP    p2pbotnet-host2:1247   mta-v8.mail.vip.mud.yahoo.com:smtp  SYN_SENT
TCP    p2pbotnet-host2:1248   md.mx.aol.com:smtp     ESTABLISHED
TCP    p2pbotnet-host2:1249   mta-v8.mail.vip.mud.yahoo.com:smtp  ESTABLISHED
```

Overall, looking at `netstat`, we could detect a lot of Peacomm connections. In our case we saw 83 Peacomm related connections in a total of 91 connections.

4 Detection

In section 3 on page 14, we have seen several characteristics that can identify the presence of Peacomm in the network. The more network administrators combine these characteristics, the more accurate they can identify a Peacomm host. The following detection methods can give a more obvious result after office hours or when a (possibly infected) production host isn't generating a lot of network traffic. Where possible, we gave a general advice on how to detect peer-to-peer botnets.

4.1 Protocol traffic

Peacomm "notices" when it has no connection to the internet, and will cease traffic moments later. The bot will however retry to poll all the known peer-to-peer nodes after 10 minutes. This is also very unusual: a lot of outgoing traffic on the network being generated after exactly 10 minutes. Although this is not a very useful characteristic by itself, it can be used when debugging a host's network traffic. It can also being used to blacklist IP addresses being polled.

Peacomm uses an unusual amount of UDP traffic in comparison with TCP traffic: normal office and home usage consist only of small amounts of UDP. The most common usage, e.g. webbrowsing, email and chat use TCP to communicate. One of the few UDP most common usages are streaming media, Network Time Protocol and DNS requests. Other peer-to-peer networks like Bittorrent, eDonkey/Emule and Direct Connect mainly use TCP.

When Peacomm connects to his peers, he sends a lot of packets in a very short amount of time, all originating from the same source port. This could also point to legit use of peer-to-peer traffic (also a lot of traffic with multiple destinations), so we cannot rely on this characteristic only. In the case of Peacomm, it uses UDP on a high numbered port in contrast with peer-to-peer traffic that uses a lot more TCP.

Same packet sizes: All the Peacomm packets are encrypted, but there are a unique property: most packets (96.2505%) are fixed size (see table 11 and table 3 on page 19).

Packet size	Occurance	Percentage
Total	32591	100%
Variable	1222	3,7495%
60 bytes	7589	23,2856%
61 bytes	140	0,4296%
65 bytes	51	0,1564%
67 bytes	23504	72,1181%
80 bytes	85	0,2608%

Table 11: Peacomm packet sizes

Each time a infected node tries to connect to the botnet, it uses the same string to connect to the botnet. This string does not change during the session, and in combination with the message type (eDonkey message type 0xa6) and it's fixed size (67 bytes), it can be identified as fairly unique.

Peacomm specific infected hosts can thus be identified by watching UDP traffic on high numbered ports with largely a packet size of 67 bytes and with message type 0xa6.
In general, watching spikes to odd numbered ports can be a way to narrow down any infected

hosts, regardless of botnet type. This can't be confused with legit use of peer-to-peer traffic because in the case of torrent traffic it doesn't show so extravagant spikes in network traffic and use known torrent ports (if configured by default). In spite of some similarities between regular peer-to-peer traffic and Peacomm, when talking about traffic on high numbered ports, we could see a more constant network flow for peer-to-peer traffic. Peer-to-peer traffic uses also a lot more variable packet sizes. We assume that future peer-to-peer botnets will show similar spikes with more or less the same packet sizes as they only use this for discovering peers. This technique isn't bulletproof but can give an indication and can be used in combination with other methods.

4.2 SMTP & MX queries

When the spam mailer has been activated, there are a lot of DNS MX requests in an very short timespan (See figure 11). Normal clients rarely do MX requests to other DNS servers then to the DNS server in their own network, because usually clients only use the mailserver in their domain or the mailserver from their Internet Service Provider.
This is not only a good way to detect Peacomm infected hosts. It can be used to detect all hosts infected by this kind of malware and engaged in spamming. This can be centralized by monitoring traffic of DNS servers and see which hosts make a lot of MX queries.

An increasing amount of virus mails. Peacomm usually travels via email attachments (as .exe). If the network administrators don't filter out virus and spam mails, they can expect to have the worm on their networks soon due to users activating the bot.

4.3 Connections

When an infected host starts spamming, he makes a lot of SMTP connections. This information can also be analyzed to detect infected hosts. This information is only related to SMTP. A inherent characteristic of peer-to-peer traffic is establishing a connection with all his peers. So hosts using legit peer-to-peer traffic will also establish a lot of connections but aren't necessarily infected.

5 Conclusion and future work

We have presented an overview of (peer-to-peer) botnets and described the evasion tactics used by their programmers. In essence, all these botnets are designed for one goal: making money. Peer-to-peer botnets use a more sophisticated method but by doing so they make a lot more connections inherent to peer-to-peer traffic.

It is in the bot writer's best interest not to propagate their code to the media because of the programmer's wish to keep it a secret and infected as many hosts as possible. Thus, counteractions can only be taken after detection and a certain period of time analyzing their structure and weaknesses. It is also hard to predict what these programmers will incorporate next in their code.

Our Peacomm case study gives a better insight in one implementation of peer-to-peer functionality used by botnets. At the moment, it is one of a kind. Looking at the test results, we have identified some unique characteristics about Peacomm, which differs from other common network traffic. Some characteristics can be used to detect peer-to-peer botnets in general. Readers can use these characteristics to help them identify possible malicious traffic.

As for SURFnet, it is important to keep up to date on the development of these bots. With every new threat, research must be done in the field of detection, and eventually must take counteractions. We expect to hear more from Peacomm in the near future and we think that other botnet families will choose for more resilient architectures. SURFnet needs to keep a close eye on future Peacomm developments. We suspect a less noisy propagation in future releases. Investigation of these releases must be done to ensure the effectiveness of the used detection methods.

When detecting new bots in the future, more research can be done in the field of analysis and detection and see if our proposal on detection is still effective against these new threats. Agobot, one of the most successful IRC-based botnets, is one of the prime candidates to increase his level of sophistication and choose for a P2P structure.

Our Peacomm case study and the advice given for detecting bots must be looked into by SURFnet. If it is feasible for SURFnet, they can use our advice to implement it in their infrastructure.

References

[1] Microsoft.com Solution Accelerators. Malware removal starter kit: How to combat malware using windows pe. Technical report, Microsoft.com, 2007. http://www.microsoft.com/downloads/details.aspx?FamilyId=6cd853ce-f349-4a18-a14f-c99b64adfbea&displaylang=en.

[2] X-Ways Software Technology AG. Winhex: Computer forensics & data recovery software, hex editor & disk editor. http://www.x-ways.net/winhex/, 2008.

[3] The Honeynet Project & Research Alliance. Know your enemy: Fast-flux service networks. Technical report, The Honeynet Project & Research Alliance, 2007.

[4] Cisco. Netflow. http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html, 2008.

[5] M. Suenaga & Mircea Ciubotariu. Symantec: Trojan.peacomm. http://www.symantec.com/securityresponse/writeup.jsp?docid=2007-011917-1403-99, 2007.

[6] Gerald Combs. Wireshark: Go deep.:. http://www.wireshark.org/, 2008.

[7] Wikipedia contributors. Chord (dht). http://en.wikipedia.org/wiki/Chord_\%28DHT\%29, 2008.

[8] Wikipedia contributors. Distributed hash tables. http://en.wikipedia.org/wiki/Distributed_hash_table, 2008.

[9] Wikipedia contributors. Fast flux. http://en.wikipedia.org/wiki/Fast_flux, 2008.

[10] Wikipedia contributors. Hash tables. http://en.wikipedia.org/wiki/Hash_table, 2008.

[11] Wikipedia contributors. Overnet / kadamlia. http://en.wikipedia.org/wiki/Overnet, 2008.

[12] Wikipedia contributors. Peer-to-peer. http://en.wikipedia.org/wiki/Peer-to-peer, 2008.

[13] Wikipedia contributors. Storm botnet. http://en.wikipedia.org/wiki/Storm_botnet, 2008.

[14] Julian B. Grizzard & Vikram Sharma & Chris Nunnery & Brent ByungHoon Kang & David Dagon. Peer-to-peer botnets: Overview and case study. Hotbots '07 workshop program, The Johns Hopkins University Applied Physics Laboratory, University of North Carolina at Charlotte, Georgia Institute of Technology, 2007. http://www.usenix.org/events/hotbots07/tech/full_papers/grizzard/grizzard_html/.

[15] Dave Dittrich & Sven Dietrich. command and control structures in malware. ;login: publication, http://www.usenix.org, 2007.

[16] DigitalNinjitsu.com. Perileyez - settings your sights on malicious code. http://digitalninjitsu.com/downloads.html, 2006.

[17] EP_X0FF and the development team. Rootkit unhooker. http://antirootkit.com/software/RootKit-Unhooker.htm, 2007.

[18] Shawn Flanning. Napster free–listen to free streaming music online:. http://free.napster.com/, 2008.

[19] S. Yang & H. Garcia-Molina. Designing a super-peer network. Proc. 19th int'l conf. data engineering, (bangalore, india), CA: IEEE Computer Society Press, 2003.

[20] Anestis Karasaridis & Brian Rexroad & David Hoeflin. Wide-scale botnet detection and characterization. Hotbots '07 workshop program, AT&T Labs, 2007.

[21] Internet Assigned Numbers Authority (IANA). Port numbers. `http://www.cs.columbia.edu/~hgs/internet/traffic.html`, 2008.

[22] IPCop.org. Ipcop.org :: The bad packets stop here!:. `http://ipcop.org/`, 2008.

[23] Gregg Keizer. Storm switches tactics third time, adds rootkit. `Stormswitchestacticsthirdtime,addsrootkit`, 2007.

[24] Reinier Schoof & Ralph Koning. Detecting peer-to-peer botnets, 2007. Research project 1 report for the Master program System and Network Engineering.

[25] Amir Lev. The elusive enemy. `http://www.secprodonline.com/articles/48471/`, 2007.

[26] Petar Maymounkov and David Mazieres. Kademlia: A peer-to-peer information system based on the xor metric. `http://www.cs.rice.edu/Conferences/IPTPS02/109.pdf`, 2008.

[27] Evan Cooke & Farnam Jahanian & Danny McPherson. The zombie roundup: Understanding, detecting, and disrupting botnets. Technical report, Electrical Engineering and Computer Science Department, University of Michigan, 2005.

[28] Microsoft.com. Malicious software removal tool. `http://www.microsoft.com/security/malwareremove/default.mspx`, 2008.

[29] Nmap. Nmap - free security scanner for network exploration & security audits. `http://nmap.org/`, 2008.

[30] Axel Bock Oliver Heckmann. The edonkey 2000 protocol. `ftp://ftp.kom.e-technik.tu-darmstadt.de/pub/papers/HB02-1-paper.pdf`, 2002.

[31] Phillip Porras, Hassen Saidi, and Vinod Yegneswaran. A multi-perspective analysis of the storm (peacomm) worm. `http://www.cyber-ta.org/pubs/StormWorm/SRITechnical-Report-10-01-Storm-Analysis.pdf`, 2007.

[32] The Honeynet Project. Slapper worm.unlock.c source le. `http://www.honeynet.org/scans/scan25/.unlock.nl.c`, 2002.

[33] Niels Provos and Thorsten Holz. *Virtual Honeypots*. Addison-Wesley, 2008.

[34] Bryce Cogswell & Mark Russinovich. Rootkitrevealer v1.71. `http://technet.microsoft.com/en-us/sysinternals/bb897445.aspx`, 2006.

[35] Henning Schulzrinne. Long-term traffic statistics. `http://www.cs.columbia.edu/~hgs/internet/traffic.html`, 2008.

[36] Sunbelt Software. Cwsandbox. `http://cwsandbox.org/?page=submit`, 2008.

[37] Andrew S. Tanenbaum & Maarten Van Steen. *Distributed Systems: Principles and Paradigms*. Pearson Prentice Hall, second edition edition, 2007. ISBN 0-13-239227-5.

[38] Joe Stewart. Threat analysis - storm worm ddos attack. `http://www.secureworks.com/research/threats/view.html?threat=storm-worm`, 2007.

[39] Symantec. Crimeware: Bots. `http://www.symantec.com/avcenter/cybercrime/bots_page1.html`, 2008.

[40] Moheeb Abu Rajab & Jay Zarfoss & Fabian Monrose & Andreas Terzis. My botnet is bigger than yours (maybe, better than yours): why size estimates remain challenging. Hotbots '07 workshop program, Computer Science Department, Johns Hopkins University, 2007. `http://www.usenix.org/events/hotbots07/tech/full_papers/rajab/rajab_html/`.

[41] uTorrent. http://www.utorrent.com/. `http://www.utorrent.com/`, 2008.

[42] Lisa Vaas. Researchers: Botnets getting beefier. `http://www.eweek.com/c/a/Security/Researchers-Botnets-Getting-Beefier/`, 2007.

[43] M. Zalewski. "i don't think i really love you". `http://seclists.org/vuln-dev/2000/May/0159.html`, 2000.

[44] David Zimmer. Sysanalyzer. `http://labs.idefense.com/software/malcode.php`, 2007.

[45] Ping Wang & Sherri Sparks & Cliff C. Zou. An advanced hybrid peer-to-peer botnet. Hotbots '07 workshop program, School of Electrical Engineering and Computer Science, University of Central Florida, 2007.

A Attachments

A.1 Peacomm fast-fluxnetwork

A.1.1 Abstract

Analysis of Peacomm's network traffic does not point to the use of fast-flux networks to obfuscate the command and controls servers. It does however use fast-flux networks to obfuscate that Peacomm itself deploys or helps deploying online shops with pharmacy products. The IP addresses in the A records for the domains are most likely part of the peacomm botnet, serving the webpages.

A.1.2 Analysis

We saw that Peacomm started spamming, because we saw a lot of DNS MX requests coming by. We then analyzed the packets and found several SMTP sessions. The bot was doing MX requests to a lot of domainnames, and querying them for open relays. Once it found one, it used it for spamming.

```
220 bay0-mc10-f9.bay0.hotmail.com Sending unsolicited commercial or bulk e-mail to Microsoft's
 computer network is prohibited. Other restrictions are found at http://privacy.msn.com/Anti-s
pam/. Violations will result in use of equipment located in California and other states. Thu,
17 Jan 2008 03:42:14 -0800

HELO exp16.studexp.os3.nl

250 bay0-mc10-f9.bay0.hotmail.com (3.5.0.22) Hello [145.100.102.186]

MAIL From:<kit@centrum.cz>

250 kit@centrum.cz....Sender OK

RCPT TO:<guaira16@msn.com>

250 guaira16@msn.com

DATA

354 Start mail input; end with <CRLF>.<CRLF>

Received: from aqhq ([159.235.71.62]) by exp16.studexp.os3.nl with Microsoft SMTPSVC(5.0.2195.67
13); Thu, 17 Jan 2008 12:42:18 +0100
Message-ID: <001001c858fe$03b8f690$3e47eb9f@aqhq>
From: <kit@centrum.cz>
To: <guaira16@msn.com>
Subject: Bring more happiness into your holiday mood
Date: Thu, 17 Jan 2008 12:42:18 +0100
MIME-Version: 1.0
Content-Type: text/plain;
.format=flowed;
.charset="iso-8859-1";
.reply-type=original
Content-Transfer-Encoding: 7bit
X-Priority: 3
X-MSMail-Priority: Normal
X-Mailer: Microsoft Outlook Express 6.00.2800.1158
X-MimeOLE: Produced By Microsoft MimeOLE V6.00.2800.1158

She will laugh at your small stick anymore
http://ilqbq.windowsoxonline.com

.
250  <001001c858fe$03b8f690$3e47eb9f@aqhq> Queued mail for delivery

QUIT
```

```
221 bay0-mc10-f9.bay0.hotmail.com Service closing transmission channel
```

When users goto the link posted in the email, they are redirected to a web site where they offer pharmacy products.

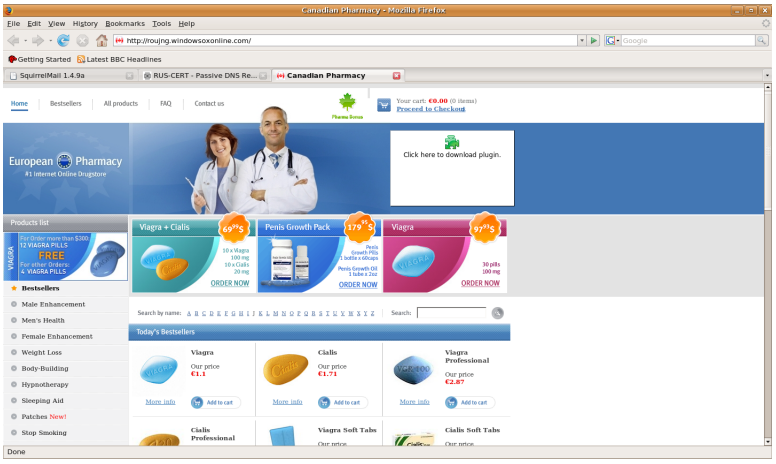

Figure 12: Peacomm spam site

We did a DIG[17] to request the A record, we got a lot of IP addresses back. It is very likely that the IP addresses listed here are also part of the botnet and serve the websites. The A records have a suspiciously low TTL (300 seconds or 5 minutes). The values (IP addresses) in the A records are also constantly changing, when the A records expires.
The nameservers also have a low TTL and there are 4 of them. The 4 nameservers have a very unusual domainname. The domain has been registered very recently at a chinese domain, and, as of writing, has not been taken offline yet.

```
;; ANSWER SECTION:
windowsoxonline.com.    300     IN      A       58.103.16.179
windowsoxonline.com.    300     IN      A       59.10.217.239
windowsoxonline.com.    300     IN      A       59.17.208.96
windowsoxonline.com.    300     IN      A       61.76.250.122
windowsoxonline.com.    300     IN      A       61.238.72.41
windowsoxonline.com.    300     IN      A       69.228.33.128
windowsoxonline.com.    300     IN      A       79.120.81.229
windowsoxonline.com.    300     IN      A       122.29.77.108
windowsoxonline.com.    300     IN      A       123.98.179.124
windowsoxonline.com.    300     IN      A       123.202.189.143
windowsoxonline.com.    300     IN      A       221.127.1.2
windowsoxonline.com.    300     IN      A       221.127.15.97
windowsoxonline.com.    300     IN      A       221.127.44.200
windowsoxonline.com.    300     IN      A       221.127.59.193
windowsoxonline.com.    300     IN      A       221.127.233.1

;; AUTHORITY SECTION:
windowsoxonline.com.    300     IN      NS      ns0.fyukbz.com.
windowsoxonline.com.    300     IN      NS      ns0.ahfywbz.com.
windowsoxonline.com.    300     IN      NS      ns0.ckjdtybz.com.
windowsoxonline.com.    300     IN      NS      ns0.uthvfybz.com.
```

[17]Domain Information Groper is a network tool, like nslookup, that queries DNS name servers.

Registrar information:

```
Domain Name: WINDOWSOXONLINE.COM
Registrar: BEIJING INNOVATIVE LINKAGE TECHNOLOGY LTD. DBA DNS.COM.CN
Whois Server: whois.dns.com.cn
Referral URL: http://www.dns.com.cn
Name Server: NS0.CKJDTYBZ.COM
Name Server: NS0.UTHVFYBZ.COM
Name Server: NS0.AHFYWBZ.COM
Name Server: NS0.FYUKBZ.COM
Status: clientTransferProhibited
Updated Date: 16-jan-2008
Creation Date: 15-jan-2008
Expiration Date: 15-jan-2009

Domain Name: FYUKBZ.COM
Registrar: BEIJING INNOVATIVE LINKAGE TECHNOLOGY LTD. DBA DNS.COM.CN
Whois Server: whois.dns.com.cn
Referral URL: http://www.dns.com.cn
Name Server: NS1.DNS.COM.CN
Name Server: NS2.DNS.COM.CN
Status: clientTransferProhibited
Updated Date: 13-jan-2008
Creation Date: 13-jan-2008
Expiration Date: 13-jan-2009
```

A.2 Sandbox

A.2.1 CWSandbox 2.0.33

```
CWSandbox Analysis report for file: dancer.exe

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Processes 1 (c:\dancer.exe MD5: [2437f5580899472e5ae20d85005d2a1f], PID 3604, User: john)
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

Trojan.Peed-45

================================================================================
DLL-Handling
================================================================================

Loaded DLL - DLL: (C:\WINDOWS\system32\ntdll.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\kernel32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\advapi32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\RPCRT4.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\gdi32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\USER32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\oleaut32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\msvcrt.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\ole32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\wsock32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\WS2_32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\WS2HELP.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\comctl32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\Wship6.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\pstorec.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\ATL.DLL)
Loaded DLL - DLL: (C:\WINDOWS\system32\Secur32.dll)
Loaded DLL - DLL: (WS2_32.dll)
Loaded DLL - DLL: (advapi32.dll)
Loaded DLL - DLL: (kernel32.dll)
Loaded DLL - DLL: (comctl32.dll)
Loaded DLL - DLL: (WININET.dll)
Loaded DLL - DLL: (DNSAPI.dll)
Loaded DLL - DLL: (KERNEL32.dll)
Loaded DLL - DLL: (USER32.dll)
Loaded DLL - DLL: (ADVAPI32.dll)
Loaded DLL - DLL: (VERSION.dll)

================================================================================
Filesystem Changes
================================================================================
Copy File: c:\dancer.exe to C:\WINDOWS\noskrnl.exe
Find File: w32tm.exe
Find File: noskrnl.exe
Open File: C:\WINDOWS\AppPatch\sysmain.sdb (OPEN_EXISTING),
(FILE_ANY_ACCESS,FILE_READ_ATTRIBUTES), (SHARE_READ),
(FILE_ATTRIBUTE_NORMAL,SECURITY_ANONYMOUS)
Open File: C:\WINDOWS\AppPatch\systest.sdb (OPEN_EXISTING),
```

```
(FILE_ANY_ACCESS,FILE_READ_ATTRIBUTES), (SHARE_READ),
(FILE_ATTRIBUTE_NORMAL,SECURITY_ANONYMOUS)
Open File: \Device\NamedPipe\ShimViewer (OPEN_EXISTING),
  (FILE_ANY_ACCESS,FILE_WRITE_ACCESS,FILE_WRITE_DATA,FILE_ADD_FILE,
  FILE_ADD_SUBDIRECTORY,FILE_APPEND_DATA,FILE_CREATE_PIPE_INSTANCE,
  FILE_WRITE_EA,FILE_WRITE_ATTRIBUTES),(),(FILE_ATTRIBUTE_NORMAL,SECURITY_ANONYMOUS)
Open File: C:\WINDOWS\system32\w32tm.exe (),
(FILE_ANY_ACCESS,FILE_READ_ACCESS,FILE_READ_DATA,FILE_LIST_DIRECTORY),
(SHARE_READ,SHARE_WRITE), (SECURITY_ANONYMOUS)
Open File: C:\WINDOWS\noskrnl.exe (), (FILE_ANY_ACCESS,FILE_READ_ACCESS,
FILE_READ_DATA,FILE_LIST_DIRECTORY), (SHARE_READ,SHARE_WRITE), (SECURITY_ANONYMOUS)

================================================================================
Registry Changes
================================================================================
Create or Open:

Registry Changes:
HKEY_CURRENT_USER\Software\Microsoft\Windows\CurrentVersion\Run\ "" = (C:\WINDOWS\noskrnl.exe)

Registry Reads:
HKEY_LOCAL_MACHINE\SYSTEM\WPA\MediaCenter\ ""

Registry Enums:

================================================================================
Process Management
================================================================================
Creates Process - Filename () CommandLine: (w32tm /config /syncfromflags:manual
/manualpeerlist:time.windows.com,time.nist.gov) Target PID: () As User: () Creation Flags: ()
Creates Process - Filename () CommandLine: (w32tm /config /update) Target PID: ()
As User: () Creation Flags: ()
Creates Process - Filename (C:\WINDOWS\noskrnl.exe) CommandLine: () Target PID: (3684)
As User: () Creation Flags: ()
Kill Process - Filename () CommandLine: () Target PID: (3604) As User: () Creation Flags: ()

================================================================================
System Info
================================================================================
Get Windows Directory

================================================================================
Virtual Memory
================================================================================
VM Protect - Target: (3604) Address: ($771C6000) Size: (4096) Protect:
(PAGE_EXECUTE_READWRITE) Allocation Type: ()
VM Protect - Target: (3604) Address: ($771C6000) Size: (4096) Protect:
(PAGE_EXECUTE_READ) Allocation Type: ()
VM Protect - Target: (3604) Address: ($771D7000) Size: (4096) Protect:
(PAGE_EXECUTE_READWRITE) Allocation Type: ()
VM Protect - Target: (3604) Address: ($771D7000) Size: (4096) Protect:
(PAGE_EXECUTE_READ) Allocation Type: ()
VM Protect - Target: (3604) Address: ($771C4000) Size: (4096) Protect:
(PAGE_EXECUTE_READWRITE) Allocation Type: ()
```

```
VM Protect - Target: (3604) Address: ($771C4000) Size: (4096) Protect:
(PAGE_EXECUTE_READ) Allocation Type: ()
VM Protect - Target: (3604) Address: ($771D6000) Size: (4096) Protect:
(PAGE_EXECUTE_READWRITE) Allocation Type: ()
VM Protect - Target: (3604) Address: ($771D6000) Size: (4096) Protect:
(PAGE_EXECUTE_READ) Allocation Type: ()
VM Protect - Target: (3604) Address: ($771C4000) Size: (4096) Protect:
(PAGE_EXECUTE_READWRITE) Allocation Type: ()
VM Protect - Target: (3604) Address: ($771C4000) Size: (4096) Protect:
(PAGE_EXECUTE_READ) Allocation Type: ()
VM Protect - Target: (3604) Address: ($771D5000) Size: (4096) Protect:
(PAGE_EXECUTE_READWRITE) Allocation Type: ()
VM Protect - Target: (3604) Address: ($771D5000) Size: (4096) Protect:
(PAGE_EXECUTE_READ) Allocation Type: ()

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Processes 2 (w32tm /config /syncfromflags:manual /manualpeerlist:time.windows.com,
time.nist.gov MD5: [], PID 3628, User: john)
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

================================================================================
DLL-Handling
================================================================================

Loaded DLL - DLL: (C:\WINDOWS\system32\ntdll.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\kernel32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\msvcrt.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\MSVCP60.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\ADVAPI32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\RPCRT4.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\USER32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\GDI32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\WS2_32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\WS2HELP.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\NETAPI32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\w32time.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\iphlpapi.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\USERENV.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\Secur32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\icmp.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\NTDSAPI.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\DNSAPI.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\WLDAP32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\SHELL32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\SHLWAPI.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\ShimEng.dll)
Loaded DLL - DLL: (C:\WINDOWS\AppPatch\AcGenral.DLL)
Loaded DLL - DLL: (C:\WINDOWS\system32\WINMM.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\ole32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\OLEAUT32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\MSACM32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\VERSION.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\UxTheme.dll)
Loaded DLL - DLL: (C:\WINDOWS\WinSxS\x86_Microsoft.Windows.
```

```
Common-Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9\)
Loaded DLL - DLL: (C:\WINDOWS\system32\comctl32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\wsock32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\Wship6.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\pstorec.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\ATL.DLL)

================================================================================
Registry Changes
================================================================================
Create or Open:

Registry Changes:
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\W32Time\Parameters\ ""
= (time.windows.com,time.nist.gov)
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\W32Time\Parameters\ "" = (NTP)

Registry Reads:

Registry Enums:

================================================================================
Process Management
================================================================================
Kill Process - Filename () CommandLine: () Target PID: (3628) As User: () Creation Flags: ()

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Processes 3 (w32tm /config /update MD5: [], PID 3648, User: john)
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

================================================================================
DLL-Handling
================================================================================

Loaded DLL - DLL: (C:\WINDOWS\system32\ntdll.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\kernel32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\msvcrt.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\MSVCP60.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\ADVAPI32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\RPCRT4.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\USER32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\GDI32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\WS2_32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\WS2HELP.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\NETAPI32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\w32time.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\iphlpapi.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\USERENV.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\Secur32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\icmp.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\NTDSAPI.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\DNSAPI.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\WLDAP32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\SHELL32.dll)
```

```
Loaded DLL - DLL: (C:\WINDOWS\system32\SHLWAPI.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\ShimEng.dll)
Loaded DLL - DLL: (C:\WINDOWS\AppPatch\AcGenral.DLL)
Loaded DLL - DLL: (C:\WINDOWS\system32\WINMM.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\ole32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\OLEAUT32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\MSACM32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\VERSION.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\UxTheme.dll)
Loaded DLL - DLL: (C:\WINDOWS\WinSxS\x86_Microsoft.Windows.
Common-Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9\)
Loaded DLL - DLL: (C:\WINDOWS\system32\comctl32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\wsock32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\Wship6.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\pstorec.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\ATL.DLL)

================================================================================
Process Management
================================================================================
Kill Process - Filename () CommandLine: () Target PID: (3648) As User: () Creation Flags: ()

================================================================================
Service Management
================================================================================
Open Service Manager - Name: (SCM) Start Type: ()
Open Service - Name: (w32time) Start Type: ()
Control Service - Name: (w32time) Display Name: () File Name: () Control: () Start Type: ()

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Processes 4 (C:\WINDOWS\noskrnl.exe MD5: [2437f5580899472e5ae20d85005d2a1f], PID 3684,
User: john)
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

Trojan.Peed-45

================================================================================
DLL-Handling
================================================================================

Loaded DLL - DLL: (C:\WINDOWS\system32\ntdll.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\kernel32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\advapi32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\RPCRT4.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\gdi32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\USER32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\oleaut32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\msvcrt.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\ole32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\wsock32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\WS2_32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\WS2HELP.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\comctl32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\Wship6.dll)
```

```
Loaded DLL - DLL: (C:\WINDOWS\system32\pstorec.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\ATL.DLL)
Loaded DLL - DLL: (C:\WINDOWS\system32\Secur32.dll)
Loaded DLL - DLL: (WS2_32.dll)
Loaded DLL - DLL: (advapi32.dll)
Loaded DLL - DLL: (kernel32.dll)
Loaded DLL - DLL: (comctl32.dll)
Loaded DLL - DLL: (WININET.dll)
Loaded DLL - DLL: (DNSAPI.dll)
Loaded DLL - DLL: (KERNEL32.dll)
Loaded DLL - DLL: (USER32.dll)
Loaded DLL - DLL: (ADVAPI32.dll)
Loaded DLL - DLL: (advapi32)
Loaded DLL - DLL: (VERSION.dll)

================================================================================
Filesystem Changes
================================================================================
Find File: netsh.exe
Move File: C:\DOCUME~1\john\LOCALS~1\Temp\\ff34ff45 to
Create File: C:\WINDOWS\noskrnl.config
Delete File: C:\WINDOWS\system32\noskrnl.sys
Open File: C:\DOCUME~1\john\LOCALS~1\Temp\\ff34ff45 (OPEN_EXISTING), (FILE_ANY_ACCESS),
(), (FILE_ATTRIBUTE_NORMAL,SECURITY_ANONYMOUS)
Open File: C:\WINDOWS\AppPatch\sysmain.sdb (OPEN_EXISTING),
(FILE_ANY_ACCESS,FILE_READ_ATTRIBUTES), (SHARE_READ), (FILE_ATTRIBUTE_NORMAL,SECURITY_ANONYMOUS)
Open File: C:\WINDOWS\AppPatch\systest.sdb (OPEN_EXISTING),
(FILE_ANY_ACCESS,FILE_READ_ATTRIBUTES), (SHARE_READ), (FILE_ATTRIBUTE_NORMAL,SECURITY_ANONYMOUS)
Open File: \Device\NamedPipe\ShimViewer (OPEN_EXISTING),
  (FILE_ANY_ACCESS,FILE_WRITE_ACCESS,FILE_WRITE_DATA,FILE_ADD_FILE,FILE_ADD_SUBDIRECTORY,
  FILE_APPEND_DATA,FILE_CREATE_PIPE_INSTANCE,FILE_WRITE_EA,FILE_WRITE_ATTRIBUTES), (),
  (FILE_ATTRIBUTE_NORMAL,SECURITY_ANONYMOUS)
Open File: C:\WINDOWS\system32\netsh.exe (),
(FILE_ANY_ACCESS,FILE_READ_ACCESS,FILE_READ_DATA,FILE_LIST_DIRECTORY),
(SHARE_READ,SHARE_WRITE), (SECURITY_ANONYMOUS)
Open File: \\.\PIPE\lsarpc (OPEN_EXISTING), (FILE_ANY_ACCESS),
(SHARE_READ,SHARE_WRITE), (SECURITY_ANONYMOUS)
Create/Open File: C:\WINDOWS\system32\noskrnl.sys (OPEN_ALWAYS), (FILE_ANY_ACCESS),
(), (FILE_ATTRIBUTE_NORMAL,SECURITY_ANONYMOUS)
Create/Open File: C:\DOCUME~1\john\LOCALS~1\Temp\\ff34ff45 (OPEN_ALWAYS), (FILE_ANY_ACCESS),
(), (FILE_ATTRIBUTE_NORMAL,SECURITY_ANONYMOUS)
Get File Attributes: C:\WINDOWS\noskrnl.config Flags: (SECURITY_ANONYMOUS)

================================================================================
INI Files
================================================================================
Read from INI file: C:\WINDOWS\noskrnl.config [local] uport =
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 00003D6C8F338A3FDD3DF3648666F55C
= 4B4061DB566300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 0100A634122F3553A046EC451061927C
= 4A82AB80221500
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 02007E238D780D25FD5511285E2E596E
= 3D6A408B14B200
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 03001E62DC533E7AF6161729A953891B
```

```
= 40E60464042D00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 0400EB5EC13599373A3D544A2D6AF94F
= 8EA7F4E2438E00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 05004710B3440F5D2117CE555A62D04A
= 59196393479C00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] D003971C313682735501391F296AFA36
= 84EF0172CC9300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] D1039A3BA822387F7570B2105E017737
= 84EF0172CC9300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] D303A41C6E67C706A066703F625B373E
= 84EF0172CC9300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] D4036157C9587D228152C4616E610008
= 84EF0172CC9300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] D5039D6AE30B9554595E5E6531235D37
= 8FD7811A84D300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] D60346467E53D16E06508441F7572B42
= 84EF0172CC9300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] D703960B65138E7FE449442EDF0B963B
= 44C67FB6625D00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] D803A909882A9F352023206D1D5E6A6E
= 84EF0172CC9300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] D9035B2C013C5346BC044A02477A342F
= 84EF0172CC9300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] DA03AE67BA162332D35C2D20F85EFE66
= 84EF0172CC9300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] DB03DD4A5222CA251753264B834AE42F
= 84EF0172CC9300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] DC038B338B262C69ED29FE4C2450FE3C
= 84EF0172CC9300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] DD03605357739D545810F00E3845347C
= 84EF0172CC9300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] DE0311189E31AE3D3D1F2903DB0B9722
= 63E47B8542EB00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 06001471521206296D099433C93EC427
= 55632406350C00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 07002D6D5B0FE3019C56B1290A564E59
= 4A8A822B0DDC00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 0800A2417153943DC23C6C5C817C4159
= 4574E13B04AE00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 0800A2417153943DC23C6C5C817C4159
= 4574E13B04AE00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 090065102D407C584C618673541AE973
= 4BB9202A801600
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 0A000025C024CD144649E4126879A11D
= 457456D62A2800
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 0B007B746275F438B455DF7F1424EF61
= 8FD7811A84D300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 0C00260EE116DB45D4169E149003862F
= 187AEEDB473500
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 0D001E289B5F9E016A4743746E17FF16
= 4CB590F3594D00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 0E0055414112C95E48217F670351914E
= 407E739D270F00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 090065102D407C584C618673541AE973
```

```
= 4BB9202A801600
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 0A000025C024CD144649E4126879A11D
= 457456D62A2800
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 0B007B746275F438B455DF7F1424EF61
= 8FD7811A84D300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 0C00260EE116DB45D4169E149003862F
= 187AEEDB473500
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 0D001E289B5F9E016A4743746E17FF16
= 4CB590F3594D00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 0E0055414112C95E48217F670351914E
= 407E739D270F00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 0F008F46AF7BEB787A30E91CA076303C
= 442D1DB8755900
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 10000A0C4D685A18CF59C847B07F5D5E
= 40E60464042D00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 11005D7EE1772F72D91A0F132026822C
= 7DBD916C674A00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 1200B94F54002C79AD26A57D067A0139
= 46101CEA33A800
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 1300F2065813514E9C362D30B85BA617
= 7A2C9606219D00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] 1400F3744A639C26693312121319EA3D
= 47CBFC6F0C1E00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] C303717D1135BC6B692D4B13141D5823
= 7D88B19A170F00
Write to INI file: C:\WINDOWS\noskrnl.config [peers] C403154B4700EF4769138E18C51E4115
= 47EB75C53FD200
Write to INI file: C:\WINDOWS\noskrnl.config [peers] C5032F1EDF2B1A35FB03306C5202462C
= 84EF0172CC9300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] DF03FC348108C37022385E75F51F0401
= 84EF0172CC9300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] E00309457E36B3344F607966B82B6221
= 84EF0172CC9300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] E103FA49BF77B65DBF4AD8265D2AE808
= 84EF0172CC9300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] E2036F14D93DF44BDA111E3BC11BE03C
= 4B29A97B092700
Write to INI file: C:\WINDOWS\noskrnl.config [peers] E303E657EE776D23512E1C3DDA4EC06E
= 7DA2500C272400
Write to INI file: C:\WINDOWS\noskrnl.config [peers] E403CA059359D450DB083163612AE721
= 182DD206612000
Write to INI file: C:\WINDOWS\noskrnl.config [peers] E5031F69AF7886391726441422114730
= 84EF0172CC9300
Write to INI file: C:\WINDOWS\noskrnl.config [peers] E603917B717D1135BC6B692D4B13141D
= 18FDFB1C809200
Write to INI file: C:\WINDOWS\noskrnl.config [peers] E7035823154B4700EF4769138E18C51E
= 46F43A0F5BAA00
Write to INI file: C:\WINDOWS\noskrnl.config [local] uport = 17097

================================================================================
Registry Changes
================================================================================
Create or Open:
```

```
Registry Changes:

Registry Reads:
HKEY_LOCAL_MACHINE\SYSTEM\WPA\MediaCenter\ ""
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Rpc\SecurityService\ ""

Registry Enums:

================================================================================
Process Management
================================================================================
Creates Process - Filename () CommandLine: (netsh firewall set allowedprogram
"C:\WINDOWS\noskrnl.exe" enable) Target PID: () As User: () Creation Flags: ()

================================================================================
Service Management
================================================================================
Open Service Manager - Name: (SCM) Start Type: ()
Open Service - Name: (noskrnl.sys) Start Type: ()
Create Service - Name: (noskrnl.sys) Display Name: (noskrnl.sys) File Name:
(C:\WINDOWS\system32\noskrnl.sys) Control: () Start Type: (SERVICE_DEMAND_START)
Start Service - Name: (noskrnl.sys) Display Name: () File Name: () Control: () Start Type: ()

================================================================================
System
================================================================================
Sleep - Milliseconds (20000)
Sleep - Milliseconds (30000)
Sleep - Milliseconds (100)
Sleep - Milliseconds (60000)

================================================================================
System Info
================================================================================
Get System Directory
Get Windows Directory

================================================================================
Threads
================================================================================
Create Thread - Target PID (3684) Thread ID (1012) Thread ID ($0040A28A)
Parameter Address ($00149690) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (916) Thread ID ($0040A28A)
Parameter Address ($001497A8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (556) Thread ID ($0040A28A)
Parameter Address ($00147868) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1128) Thread ID ($0040A28A)
Parameter Address ($001930B8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1124) Thread ID ($0040A28A)
Parameter Address ($001A3FF8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (512) Thread ID ($0040A28A)
Parameter Address ($00153A48) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (504) Thread ID ($0040A28A)
Parameter Address ($00153AF0) Creation Flags ()
```

```
Create Thread - Target PID (3684) Thread ID (1180) Thread ID ($0040A28A)
Parameter Address ($00153B98) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (208) Thread ID ($0040A28A)
Parameter Address ($00153C40) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (212) Thread ID ($0040A28A)
Parameter Address ($00153CE8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (620) Thread ID ($0040A28A)
Parameter Address ($00153D60) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (664) Thread ID ($0040A28A)
Parameter Address ($00153E00) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1444) Thread ID ($0040A28A)
Parameter Address ($00153EA0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (944) Thread ID ($0040A28A)
Parameter Address ($00153F70) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1832) Thread ID ($0040A28A)
Parameter Address ($00154010) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1836) Thread ID ($0040A28A)
Parameter Address ($001540B0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1840) Thread ID ($0040A28A)
Parameter Address ($00154150) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1844) Thread ID ($0040A28A)
Parameter Address ($001541F0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1848) Thread ID ($0040A28A)
Parameter Address ($00154290) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1852) Thread ID ($0040A28A)
Parameter Address ($00154330) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1856) Thread ID ($0040A28A)
Parameter Address ($00154400) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1860) Thread ID ($0040A28A)
Parameter Address ($001544A0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1792) Thread ID ($0040A28A)
Parameter Address ($00154570) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1884) Thread ID ($0040A28A)
Parameter Address ($00153A48) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1968) Thread ID ($0040A28A)
Parameter Address ($001A3FF8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1976) Thread ID ($0040A28A)
Parameter Address ($001930B8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1720) Thread ID ($0040A28A)
Parameter Address ($00147868) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1072) Thread ID ($0040A28A)
Parameter Address ($001546D8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2052) Thread ID ($0040A28A)
Parameter Address ($00154570) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2056) Thread ID ($0040A28A)
Parameter Address ($001544A0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2060) Thread ID ($0040A28A)
Parameter Address ($00154400) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2068) Thread ID ($0040A28A)
Parameter Address ($00154330) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2072) Thread ID ($0040A28A)
Parameter Address ($00154290) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2076) Thread ID ($0040A28A)
Parameter Address ($00154150) Creation Flags ()
```

```
Create Thread - Target PID (3684) Thread ID (2080) Thread ID ($0040A28A)
Parameter Address ($001541F0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2116) Thread ID ($0040A28A)
Parameter Address ($001540B0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2124) Thread ID ($0040A28A)
Parameter Address ($00154010) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2132) Thread ID ($0040A28A)
Parameter Address ($00153F70) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2136) Thread ID ($0040A28A)
Parameter Address ($00153EA0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2140) Thread ID ($0040A28A)
Parameter Address ($00153E00) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2148) Thread ID ($0040A28A)
Parameter Address ($00153D60) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1276) Thread ID ($0040A28A)
Parameter Address ($00153CE8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2160) Thread ID ($0040A28A)
Parameter Address ($00153C40) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2164) Thread ID ($0040A28A)
Parameter Address ($00153B98) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2168) Thread ID ($0040A28A)
Parameter Address ($00153A48) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2184) Thread ID ($0040A28A)
Parameter Address ($001546D8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2188) Thread ID ($0040A28A)
Parameter Address ($00147868) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2192) Thread ID ($0040A28A)
Parameter Address ($001930B8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2196) Thread ID ($0040A28A)
Parameter Address ($001A3FF8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2200) Thread ID ($0040A28A)
Parameter Address ($00153C40) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2204) Thread ID ($0040A28A)
Parameter Address ($00153B98) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2208) Thread ID ($0040A28A)
Parameter Address ($00153CE8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2212) Thread ID ($0040A28A)
Parameter Address ($00153D60) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2216) Thread ID ($0040A28A)
Parameter Address ($00153E00) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2220) Thread ID ($0040A28A)
Parameter Address ($00153EA0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2224) Thread ID ($0040A28A)
Parameter Address ($00154010) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2228) Thread ID ($0040A28A)
Parameter Address ($00153F70) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2240) Thread ID ($0040A28A)
Parameter Address ($001540B0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2244) Thread ID ($0040A28A)
Parameter Address ($001541F0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2248) Thread ID ($0040A28A)
Parameter Address ($00154150) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2256) Thread ID ($0040A28A)
Parameter Address ($00154290) Creation Flags ()
```

```
Create Thread - Target PID (3684) Thread ID (2272) Thread ID ($0040A28A)
Parameter Address ($00154330) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2276) Thread ID ($0040A28A)
Parameter Address ($00154400) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2280) Thread ID ($0040A28A)
Parameter Address ($001544A0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2284) Thread ID ($0040A28A)
Parameter Address ($00154570) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2288) Thread ID ($0040A28A)
Parameter Address ($00153A48) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2300) Thread ID ($0040A28A)
Parameter Address ($001A3FF8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2304) Thread ID ($0040A28A)
Parameter Address ($001930B8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2308) Thread ID ($0040A28A)
Parameter Address ($00147868) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2312) Thread ID ($0040A28A)
Parameter Address ($001546D8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2316) Thread ID ($0040A28A)
Parameter Address ($00154570) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2320) Thread ID ($0040A28A)
Parameter Address ($001544A0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2324) Thread ID ($0040A28A)
Parameter Address ($00154400) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2328) Thread ID ($0040A28A)
Parameter Address ($00154290) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2332) Thread ID ($0040A28A)
Parameter Address ($00154330) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2336) Thread ID ($0040A28A)
Parameter Address ($00154150) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2340) Thread ID ($0040A28A)
Parameter Address ($001541F0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2344) Thread ID ($0040A28A)
Parameter Address ($001540B0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2348) Thread ID ($0040A28A)
Parameter Address ($00153F70) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2352) Thread ID ($0040A28A)
Parameter Address ($00154010) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2356) Thread ID ($0040A28A)
Parameter Address ($00153EA0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2360) Thread ID ($0040A28A)
Parameter Address ($00153E00) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2364) Thread ID ($0040A28A)
Parameter Address ($00153D60) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2368) Thread ID ($0040A28A)
Parameter Address ($00153CE8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2372) Thread ID ($0040A28A)
Parameter Address ($00153B98) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2376) Thread ID ($0040A28A)
Parameter Address ($00153C40) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2380) Thread ID ($0040A28A)
Parameter Address ($00153A48) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2384) Thread ID ($0040A28A)
Parameter Address ($00153EA0) Creation Flags ()
```

```
Create Thread - Target PID (3684) Thread ID (192) Thread ID ($0040A28A)
Parameter Address ($00147868) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2492) Thread ID ($0040A28A)
Parameter Address ($001930B8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2504) Thread ID ($0040A28A)
Parameter Address ($001A3FF8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2452) Thread ID ($0040A28A)
Parameter Address ($00153AF0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2528) Thread ID ($0040A28A)
Parameter Address ($00153E00) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2532) Thread ID ($0040A28A)
Parameter Address ($00154010) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2536) Thread ID ($0040A28A)
Parameter Address ($00153F70) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2540) Thread ID ($0040A28A)
Parameter Address ($00153CE8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2548) Thread ID ($0040A28A)
Parameter Address ($00153D60) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2552) Thread ID ($0040A28A)
Parameter Address ($001540B0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2592) Thread ID ($0040A28A)
Parameter Address ($001541F0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2596) Thread ID ($0040A28A)
Parameter Address ($00154150) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2600) Thread ID ($0040A28A)
Parameter Address ($00154330) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2604) Thread ID ($0040A28A)
Parameter Address ($00154290) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1940) Thread ID ($0040A28A)
Parameter Address ($001544A0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1612) Thread ID ($0040A28A)
Parameter Address ($00154400) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2620) Thread ID ($0040A28A)
Parameter Address ($00154570) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1904) Thread ID ($0040A28A)
Parameter Address ($00153C40) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1900) Thread ID ($0040A28A)
Parameter Address ($00153B98) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2640) Thread ID ($0040A28A)
Parameter Address ($001A3FF8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2612) Thread ID ($0040A28A)
Parameter Address ($00153A48) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2644) Thread ID ($0040A28A)
Parameter Address ($00153EA0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2668) Thread ID ($0040A28A)
Parameter Address ($00153AF0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2004) Thread ID ($0040A28A)
Parameter Address ($001930B8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2000) Thread ID ($0040A28A)
Parameter Address ($00147868) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1996) Thread ID ($0040A28A)
Parameter Address ($001546D8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (752) Thread ID ($0040A28A)
Parameter Address ($00154400) Creation Flags ()
```

```
Create Thread - Target PID (3684) Thread ID (1660) Thread ID ($0040A28A)
Parameter Address ($00154570) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1664) Thread ID ($0040A28A)
Parameter Address ($001544A0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1708) Thread ID ($0040A28A)
Parameter Address ($00154290) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2488) Thread ID ($0040A28A)
Parameter Address ($00154330) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2484) Thread ID ($0040A28A)
Parameter Address ($00154150) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2688) Thread ID ($0040A28A)
Parameter Address ($001541F0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2684) Thread ID ($0040A28A)
Parameter Address ($001540B0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2480) Thread ID ($0040A28A)
Parameter Address ($00153D60) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2476) Thread ID ($0040A28A)
Parameter Address ($00153F70) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1716) Thread ID ($0040A28A)
Parameter Address ($00153CE8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1668) Thread ID ($0040A28A)
Parameter Address ($00154010) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1880) Thread ID ($0040A28A)
Parameter Address ($00153C40) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2692) Thread ID ($0040A28A)
Parameter Address ($00153B98) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1816) Thread ID ($0040A28A)
Parameter Address ($00153A48) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (440) Thread ID ($0040A28A)
Parameter Address ($001A3FF8) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (2696) Thread ID ($0040A28A)
Parameter Address ($00153EA0) Creation Flags ()
Create Thread - Target PID (3684) Thread ID (1084) Thread ID ($0040A28A)
Parameter Address ($00147868) Creation Flags ()

================================================================================
Virtual Memory
================================================================================
VM Protect - Target: (3684) Address: ($771C6000) Size: (4096)
Protect: (PAGE_EXECUTE_READWRITE) Allocation Type: ()
VM Protect - Target: (3684) Address: ($771C6000) Size: (4096)
Protect: (PAGE_EXECUTE_READ) Allocation Type: ()
VM Protect - Target: (3684) Address: ($771D7000) Size: (4096)
Protect: (PAGE_EXECUTE_READWRITE) Allocation Type: ()
VM Protect - Target: (3684) Address: ($771D7000) Size: (4096)
Protect: (PAGE_EXECUTE_READ) Allocation Type: ()
VM Protect - Target: (3684) Address: ($771C4000) Size: (4096)
Protect: (PAGE_EXECUTE_READWRITE) Allocation Type: ()
VM Protect - Target: (3684) Address: ($771C4000) Size: (4096)
Protect: (PAGE_EXECUTE_READ) Allocation Type: ()
VM Protect - Target: (3684) Address: ($771D6000) Size: (4096)
Protect: (PAGE_EXECUTE_READWRITE) Allocation Type: ()
VM Protect - Target: (3684) Address: ($771D6000) Size: (4096)
Protect: (PAGE_EXECUTE_READ) Allocation Type: ()
```

```
VM Protect - Target: (3684) Address: ($771C4000) Size: (4096)
Protect: (PAGE_EXECUTE_READWRITE) Allocation Type: ()
VM Protect - Target: (3684) Address: ($771C4000) Size: (4096)
Protect: (PAGE_EXECUTE_READ) Allocation Type: ()
VM Protect - Target: (3684) Address: ($771D5000) Size: (4096)
Protect: (PAGE_EXECUTE_READWRITE) Allocation Type: ()
VM Protect - Target: (3684) Address: ($771D5000) Size: (4096)
Protect: (PAGE_EXECUTE_READ) Allocation Type: ()

================================================================================
Winsock
================================================================================

Opening Listening TCP Connection - Local Port: 14228 -
Connection Established: 0 - Socket: 1368

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Processes 5 (services.exe MD5: [], PID 940, User: SYSTEM)
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

================================================================================
Service Management
================================================================================
Load Driver - Name: (\Registry\Machine\System\CurrentControlSet\Services\noskrnl.sys)
File Name: ()

@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@
Processes 6 (netsh firewall set allowedprogram C:\WINDOWS\noskrnl.exe enable MD5: [],
PID 3736, User: john)
@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@@

================================================================================
COM
================================================================================
COM Create Instance: C:\WINDOWS\system32\wbem\wbemprox.dll, ProgID: (),
Interface ID: ({DC12A687-737F-11CF-884D-00AA004B2E24})
COM Create Instance: C:\WINDOWS\system32\hnetcfg.dll, ProgID: (HNetCfg.FwMgr),
Interface ID: ({F7898AF5-CAC4-4632-A2EC-DA06E5111AF2})
COM Create Instance: C:\WINDOWS\system32\hnetcfg.dll, ProgID: (HNetCfg.FwAuthorizedApplication),
Interface ID: ({B5E64FFA-C2C5-444E-A301-FB5E00018050})
COM Get Class Object: C:\WINDOWS\system32\wbem\wbemsvc.dll,
Interface ID: ({D5F569D0-593B-101A-B569-08002B2DBF7A})

================================================================================
DLL-Handling
================================================================================

Loaded DLL - DLL: (C:\WINDOWS\system32\ntdll.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\kernel32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\msvcrt.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\ADVAPI32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\RPCRT4.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\MPRAPI.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\ACTIVEDS.dll)
```

```
Loaded DLL - DLL: (C:\WINDOWS\system32\adsldpc.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\NETAPI32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\WLDAP32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\USER32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\GDI32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\ATL.DLL)
Loaded DLL - DLL: (C:\WINDOWS\system32\ole32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\OLEAUT32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\rtutils.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\SAMLIB.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\SETUPAPI.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\RASAPI32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\rasman.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\WS2_32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\WS2HELP.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\TAPI32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\SHLWAPI.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\WINMM.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\iphlpapi.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\ShimEng.dll)
Loaded DLL - DLL: (C:\WINDOWS\AppPatch\AcGenral.DLL)
Loaded DLL - DLL: (C:\WINDOWS\system32\MSACM32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\VERSION.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\SHELL32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\USERENV.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\UxTheme.dll)
Loaded DLL - DLL: (C:\WINDOWS\WinSxS\x86_Microsoft.Windows.
Common-Controls_6595b64144ccf1df_6.0.2600.2180_x-ww_a84f1ff9\)
Loaded DLL - DLL: (C:\WINDOWS\system32\comctl32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\wsock32.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\Wship6.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\pstorec.dll)
Loaded DLL - DLL: (C:\WINDOWS\system32\Secur32.dll)
Loaded DLL - DLL: (IPV6MON.DLL)
Loaded DLL - DLL: (IPMONTR.DLL)
Loaded DLL - DLL: (comctl32.dll)
Loaded DLL - DLL: (IFMON.DLL)
Loaded DLL - DLL: (IPPROMON.DLL)
Loaded DLL - DLL: (RASMONTR.DLL)
Loaded DLL - DLL: (IPXMONTR.DLL)
Loaded DLL - DLL: (IPXPROMN.DLL)
Loaded DLL - DLL: (DGNET.DLL)
Loaded DLL - DLL: (netsh.exe)
Loaded DLL - DLL: (uxtheme.dll)
Loaded DLL - DLL: (HNETMON.DLL)
Loaded DLL - DLL: (FWCFG.DLL)
Loaded DLL - DLL: (OLE32)
Loaded DLL - DLL: (OLEAUT32.dll)
Loaded DLL - DLL: (SHLWAPI.dll)

================================================================================
Filesystem Changes
================================================================================
Open File: \\.\PIPE\lsarpc (OPEN_EXISTING), (FILE_ANY_ACCESS), (SHARE_READ,SHARE_WRITE),
```

```
(SECURITY_ANONYMOUS)
Open File: C:\WINDOWS\noskrnl.exe (OPEN_EXISTING), (FILE_ANY_ACCESS,FILE_READ_ATTRIBUTES),
(), (FILE_ATTRIBUTE_NORMAL,SECURITY_ANONYMOUS)
Get File Attributes: C:\WINDOWS\Registration Flags: (SECURITY_ANONYMOUS)
Get File Attributes: C:\WINDOWS\system32\WBEM\Logs\ Flags: (SECURITY_ANONYMOUS)

================================================================================
Mutex Changes
================================================================================
Creates Mutex: CTF.LBES.MutexDefaultS-1-5-21-842925246-1965331169-725345543-1003
Creates Mutex: CTF.Compart.MutexDefaultS-1-5-21-842925246-1965331169-725345543-1003
Creates Mutex: CTF.Asm.MutexDefaultS-1-5-21-842925246-1965331169-725345543-1003
Creates Mutex: CTF.Layouts.MutexDefaultS-1-5-21-842925246-1965331169-725345543-1003
Creates Mutex: CTF.TMD.MutexDefaultS-1-5-21-842925246-1965331169-725345543-1003
Creates Mutex: CTF.TimListCache.FMPDefaultS-1-5-21-842925246-1965331169-725345543-
1003MUTEX.DefaultS-1-5-21-84292524

================================================================================
Registry Changes
================================================================================
Create or Open:

Registry Changes:
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy
\StandardProfile\AuthorizedApplications\List\ "" = (C:\WINDOWS\noskrnl.exe:*:Enabled:enable)

Registry Reads:
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Windows NT\CurrentVersion\ ""
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\CIMOM\ ""
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\CIMOM\ ""
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\CIMOM\ ""
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\CTF\SystemShared\ ""
HKEY_CURRENT_USER\Keyboard Layout\Toggle\ ""
HKEY_CURRENT_USER\Keyboard Layout\Toggle\ ""
HKEY_CURRENT_USER\Keyboard Layout\Toggle\ ""
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\CTF\ ""
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\CIMOM\ ""
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\CIMOM\ ""
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\CIMOM\ ""
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\CIMOM\ ""
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\CIMOM\ ""
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\WBEM\CIMOM\ ""
HKEY_LOCAL_MACHINE\SOFTWARE\Microsoft\Rpc\SecurityService\ ""
HKEY_LOCAL_MACHINE\SYSTEM\ControlSet001\Services\SharedAccess\Parameters\FirewallPolicy
\StandardProfile\AuthorizedApplications\List\ ""

Registry Enums:

================================================================================
Process Management
================================================================================
Kill Process - Filename () CommandLine: () Target PID: (3736) As User: () Creation Flags: ()

================================================================================
```

```
System
================================================================================
Sleep - Milliseconds (60000)
Sleep - Milliseconds (0)

================================================================================
System Info
================================================================================
Get System Directory
Get Computer Name

================================================================================
Threads
================================================================================
Create Thread - Target PID (3736) Thread ID (3768) Thread ID ($77E76BF0)
Parameter Address ($0016C850) Creation Flags ()
Create Thread - Target PID (3736) Thread ID (3772) Thread ID ($774F319A)
Parameter Address ($0016EF50) Creation Flags ()
Create Thread - Target PID (3736) Thread ID (3776) Thread ID ($77E76BF0)
Parameter Address ($0016F720) Creation Flags ()

================================================================================
Virtual Memory
================================================================================
VM Protect - Target: (3736) Address: ($71A74000) Size: (4096)
Protect: (PAGE_EXECUTE_READWRITE) Allocation Type: ()
VM Protect - Target: (3736) Address: ($71A74000) Size: (4096)
Protect: (PAGE_EXECUTE_READ) Allocation Type: ()
VM Protect - Target: (3736) Address: ($71A57000) Size: (4096)
Protect: (PAGE_EXECUTE_READWRITE) Allocation Type: ()
VM Protect - Target: (3736) Address: ($71A57000) Size: (4096)
Protect: (PAGE_EXECUTE_READ) Allocation Type: ()
VM Protect - Target: (3736) Address: ($71A6E000) Size: (4096)
Protect: (PAGE_EXECUTE_READWRITE) Allocation Type: ()
VM Protect - Target: (3736) Address: ($71A6E000) Size: (4096)
Protect: (PAGE_EXECUTE_READ) Allocation Type: ()
VM Protect - Target: (3736) Address: ($71A6F000) Size: (4096)
Protect: (PAGE_EXECUTE_READWRITE) Allocation Type: ()
VM Protect - Target: (3736) Address: ($71A6F000) Size: (4096)
Protect: (PAGE_EXECUTE_READ) Allocation Type: ()
VM Protect - Target: (3736) Address: ($71A6F000) Size: (4096)
Protect: (PAGE_EXECUTE_READWRITE) Allocation Type: ()
VM Protect - Target: (3736) Address: ($71A6F000) Size: (4096)
Protect: (PAGE_EXECUTE_READ) Allocation Type: ()

================================================================================
Window
================================================================================
Enum Windows

Report generated at 1/17/2008 2:11:00 PM with CWSandbox Version 2.0.33
This analysis was created by the CWSandbox Copyright  2006 Carsten Willems
Copyright  1996-2006 Sunbelt Software. All rights reserved.
```

A.2.2 Virus Total Scan 1.0.0

Scan Engine	Version	Signature Version	Result
AhnLab-V3	2008.1.17.11	20080117	Win-Trojan/MalPatched.Gen
AntiVir	7.6.0.48	20080117	WORM/Zhelatin.Gen
Authentium	4.93.8	20080117	Possibly a new variant of W32/STZ_like!Generic
Avast	4.7.1098.0	20080116	Win32:Tibs-BMR
AVG	7.5.0.516	20080117	Packed.Tibs
BitDefender	7.2	20080117	Trojan.Peed.INS
CAT-QuickHeal	9.00	20080116	Win32.Packed.Tibs.dn
ClamAV	0.91.2	20080117	Trojan.Peed-45
DrWeb	4.44.0.09170	20080117	Trojan.Packed.206
eSafe	7.0.15.0	20080116	Win32.Tibs.dn
eTrust-Vet	31.3.5465	20080117	Win32/Sintun.AN
Ewido	4.0	20080117	Trojan.Tibs.dn
F-Prot	4.4.2.54	20080116	W32/STZ_like!Generic
F-Secure	6.70.13260.0	20080117	Packed.Win32.Tibs.dn
FileAdvisor	1	20080117	OK
Fortinet	3.14.0.0	20080117	W32/Tibs.DN!tr
Ikarus	T3.1.1.20	20080117	Trojan-Downloader.Win32.Tibs.pf
Kaspersky	7.0.0.125	20080117	Packed.Win32.Tibs.dn
McAfee	5210	20080117	Tibs-Packed
Microsoft	1.3109	20080117	Trojan:Win32/Tibs.EV
NOD32v2	2801	20080117	probably unknown NewHeur_PE virus
Norman	5.80.02	20080117	W32/Tibs.BBTE
Panda	9.0.0.4	20080117	Adware/Adsmart
Prevx1	V2	20080117	OK
Rising	20.27.31.00	20080117	Trojan.DL.Win32.Tibs.jds
Sophos	4.24.0	20080117	Mal/Dorf-F
Sunbelt	2.2.907.0	20080117	OK
Symantec	10	20080117	Trojan.Peacomm.D
TheHacker	6.2.9.189	20080117	OK
VBA32	3.12.2.5	20080115	OK
VirusBuster	4.3.26:9	20080116	Trojan.Tibs.Gen!Pac.132
Webwasher-Gateway	6.0.1	20080117	Worm.Zhelatin.Gen