NTNU
Norwegian University of
Science and Technology

Storm Worm: A P2P Botnet

Nelly Marylise Mukamurenzi

Master of Science in Communication Technology
Submission date: February 2008
Supervisor: Svein Johan Knapskog, ITEM
Co-supervisor: Andre Aarnes, Kripos

Norwegian University of Science and Technology
Department of Telematics

Problem Description

The student will continue an ongoing research activity aimed at studying malware and malicious activity on the Internet. The student will further develop the Honeypot setup, perform experiments for data collection, as well as perform analysis of security relevant incidents. The aim of the project is to study the new upper level protocols botnets and peer-to-peer botnets, and research possible detection/prevention solutions, and potentially a method to collect evidence for criminal investigations of botnets attacks.
The project is given in cooperation with the High Tech Crime Division at the National Criminal Investigation Service (Kripos).

Assignment given: 01. October 2007
Supervisor: Svein Johan Knapskog, ITEM

Abstract

According to [1], Storm worm will be the second biggest cyber threat of 2008. Reverse engineering Storm worm has proved to be anything, but trivial and many researchers are working on detection, mitigation and protection methods.

So far the studies have focused on finding out what this malware is and how it operates. The success of Storm worm is partially due to the lack of security awareness in the average computer user. The other part of its success is the use of state-of-the-art technologies and a reputation for aggressiveness. The latter part has been the focus of the papers presented in the literature review.

The purpose of this thesis is focused on the effects of Storm worm on private or corporate computers with little to average security functions. Most of the Storm botnet is comprised of such computers. Since this thesis is in cooperation with the High Tech Crime Divistion at the Norwegian Criminal Investigation Service (Kripos), there is an interest in gathering forensic evidence for court.

In this thesis, the research so far on Storm worm is presented and compared. From the collective results and findings, a theoretical honeynet set-up is presented. Out of all the information gathered and the observations on Storm's behaviour, security measures are suggested for corporate and private computer users. For evidence collection, there are suggestions on where to start and what to look for, but because of time constraints, it is mostly theoretical.

Acknowledgements

My thanks go to:

Professor Svein Knapskog and André Årnes for their guidance and academic help.

Pål Sturla Sæther and Asbjørn Karstensen for the technical help with setting up the honeynet.

Thorsten Holz from the Honeynet Project for answering my endless questions about Storm Worm and honeynets.

Phillip Porras from SRI International for providing me with the Storm binaries.

Jan Tore Soerensen, Magdalena Dessoulavy, Simen Sandberg for their help with various aspects of this thesis.

Jon Smedsrud for answering my questions.

Kjetil Haslum for taking his time to help me figure things out.

My family and friends for putting up with my stress-induced neurosis and their moral support.

Trondheim, February 2008

Nelly Marylise Mukamurenzi

Contents

List of Figures

Chapter 1

Introduction

In this thesis, I'm mainly concerned with P2P (Peer-to-Peer) botnets, their architectures, weaknesses, strengths and operations. As a case study, I will do an in-depth analysis of Storm Worm. I will also attempt to find possible means to either prevent or quickly detect a storm worm infection and gather information for police investigations.

1.1 Motivation

Botnets first started as tools to facilitate automation on the Internet [2]. Unfortunately, as with most things, cyber criminals saw the potential of profit and started using them for various crimes such as Distributed Denial of Service (DDoS), spamming, identity theft, password stealing, phishing, etc. The main advantages of using botnets to commit cybercrimes are: - It is difficult to track the responsible parties. - The large number of machines make DDoS attacks more efficient. - They are easy to deploy and use.

The botmasters learned and they found new ways to hide. To thwart discovery, they learned to use new technologies to their advantage. This led to botnets based on more complex protocols, decentralised architectures and encryption. So far, they have proved impossible to track or dismantle and they keep getting more and more dangerous. They are much better organised and are now being used for much more serious crimes than just spam. They are now used for terrorism, blackmail, industrial espionage and various acts of vandalism such as shutting down a whole country's Internet infrastructure [3].

There have been many successful attempts to shut down botnets by the Dutch police [4] and the Norwegian telecommunications company Telenor [5] among

others. These were IRC botnets with centralised Command and Control (C&C) servers. C&C messages could be tracked back to their origin and all that was needed was to dismantle the C&C server and the botnet would be effectively destroyed.

The threat paused by botnets should not be underestimated, they are highly adaptable weapons and quite dangerous. The SANS Institute has named botnets, particularly the new advanced ones such as Storm Worm, the second greatest "cyber security menace for 2008" [1].

There are many projects working on cyber crimes including crimes using botnets. One of them is the ITU CYB (International Telecommunications Union ICT Applications and Cybersecurity) and they now have a project on botnet mitigation ITU Botnet Mitigation Toolkit [6]. Another project specifically dedicated to studying Storm Worm has just started (since January, 2008) by the Honeynet Project [7].

This thesis is written in cooperation with the the High Tech Crime Divistion at the Norwegian Criminal Investigation Service (Kripos). It is one of many projects exploring the topic of botnets and the use of honeynets to learn more about them and hopefully find some weaknesses that can be used to stop them.

1.2 Description

While the IRC (Internet Relay Chat) botnets are utilising methods like fast-flux DNS to escape detection of their C&C (and potentially dismantling), others have been looking towards a whole other structure altogether. P2P botnets may be a new trend at the moment, but they are slowly increasing in popularity, mainly because so far they have proven impossible to track and take down. They are based on P2P protocols and have a completely decentralised architecture.

In early 2007, a new worm known as Storm Worm hit the net, it spread using a mixture of social engineering and exploiting vulnerabilities in Windows XP and Windows 2000. There are two main interesting facts about Storm Worm that make it so unique; the first is that it is the first completely P2P botnet and the second is that it aggressively attacks anyone who attempts to reverse engineer it. In this thesis, I will study P2P botnets and in particular Storm Worm which is quickly becoming the most dangerous botnet there is. I will first give a brief account of the botnets' evolution from IRC to P2P and what led to it. Further I will focus on P2P botnets, their architecture, weaknesses and

various forms. I will also discuss Storm Worm more closely and take a look at the research that has been done so far and analyses by some industry experts.

As this is a project for KRIPOS, my main goal is to find out how quickly Storm Worm can be detected before it compromises a system, if there are any preventative or defensive measures, how the average person can avoid getting infected and if any useful forensic information can be gathered after an attack. I will present methods and information on where to look when gathering evidence in the case of a crime investigation.

1.3 Scope

In this thesis, I am mainly concerned with P2P botnets, their architectures, weaknesses, strengths and operations. As a case study, I will do an in-depth analysis of Storm Worm. I will present a theoretical honeynet set-up, where a storm worm sample can be run in a controlled environment to collect data for analysis of its behaviour. I will also attempt to find possible means to either prevent or quickly detect a storm worm infection and gather information in the context of police investigations and digital forensics.

1.4 Structure of this report

This thesis is organised as follows:

Chapter 2: *Theory* is the background chapter that presents the evolution of botnets from centralised structures to today's P2P structures. This chapter also briefly introduces Storm Worm.

Chapter 3: *Literature Review* presents previous research on Storm Worm and the advances made in its detection and prevention. It also presents some of the developped tools and other techniques that are still under research. This section is fairly limited due to limited published in-depth analyses of Storm worm, mainly because it's a fairly new malware.

Chapter 4: *Security measures against Storm Worm* gives a brief overview of suggested security measures for preventing and detecting Storm Worm infections. This is based on the known properties of Storm Worm so far.

Chapter 5: *Honeynet Experiment* presents the Honeynet experiment set-up. First the set-up and the tools used are presented.

Chapter 6: *Data Collection and Analysis* In this section, the results of the set-up are presented as well as the evidential indications of an infection, an impending attack and the attacks themselves.

Chapter 6: *Further work* discusses possible future research areas connected to this thesis.

Chapter 7: *Conclusion* is the conclusion of the work of this thesis and the gained insight from it.

Appendices contain all the data collected from the Honeynet experiment and any other files or figures relegated to this part due to space limitations.

Chapter 2

Theory

A botnet is a collection of computers that have been infected with a software code such as a worm to allow someone (a bot master) to control them remotely without having to log on to their operating system. It can be used for innocent (Eggdrop bot by Jeff Fisher, 1993) or malicious purposes. [8] In this thesis, I am only looking at the malicious uses of botnets. This thesis is a continuation of previous research on the topic of botnets, it concentrates mainly on the newly emerging types of botnets. Below is a short history of botnets and how they have evolved through time to make it more difficult to track them or take them down.

2.1 A history of botnets

Botnets were originally controlled by a botmaster who used a Command and Control (C&C) server to communicate with the rest of the botnet. Most of these botnets were based on the IRC (Internet Relay Chat) infrastructure in which the IRC server would also be the C&C server. As we will see, there can be several C&C servers and the architectures became more and more distributed. This was caused by a need for redundancy due to the vulnerability of a centralised architecture.

The malicious uses of a botnet are most commonly for e-mail spamming, data theft/spying and DDoS (Distributed Denial of Service) attacks. They are used to avoid identification, to launch large-scale attacks and it is slowly turning into a profitable business to sell that kind of computing power to whoever is willing to pay for it.

As botnets keep getting more and more sophisticated, some have moved from IRC to more complex protocols such as HTTP (HyperText Transfer Protocol),

FTP (File Transfer Protocol) and various P2P protocols. While IRC botnets remain a majority, most of them have also evolved and become more sophisticated by using encryption and/or other protocols for communication/propagation/C&C.

Botmasters exploit vulnerabilities on the target computers that allow remote control of the computer. There are many such vulnerabilities especially on personal computers (PC). The average home PC is the common target for botnets because they are usually not well secured, if secured at all. Most people are not aware of the dangers from the Internet or what they can do about them. Due to the prevalence of broadband connections that are left on all the time, the botmaster has several active bots in the network at any given time.

2.2 IRC botnets

Originally botnets were based on IRC and many of them still are. A botherder sets up an IRC server and clients connect to it. The server acts as the C&C from where the bots get their orders. All communication is based on the IRC protocol.

While IRC is a practical technology to use for botmasters, there is a problem: IRC botnets have a centralised control and can therefore be taken down if the C&C server is located. This weakness has become more and more of a risk to the botmasters, with massive botnet take-downs by Telenor [5] and the Dutch police [4]. This has caused botmasters to adapt and find ways of protecting their C&C with more sophisticated protocols and security technologies.

The figure above 2.1 is of an IRC botnet with multiple servers. While it is more secure and redundant than a single server, it is still centralised. Moreover, although it is hard to take down the entire network, the bots that are hardcoded with the address of the now inactive servers can no longer connect to the botnet. The botmaster can work around this by hardcoding the bots with alternate server addresses. The downside is that if one bot is discovered to be compromised, it will lead to more servers.

The second attempt to thwart the efforts of bothunters (borrowing the term "bothunter" from SRI [10]) was to use DNS multihoming. The bots are now hardcoded with a domain name for the C&C. When they try to connect to that name and DNS looks up its IP address and connects to it. If the IP address is inactive (has been discovered for example), another IP (Internet Protocol) address is used. The name is still the same, but the address changes. Now

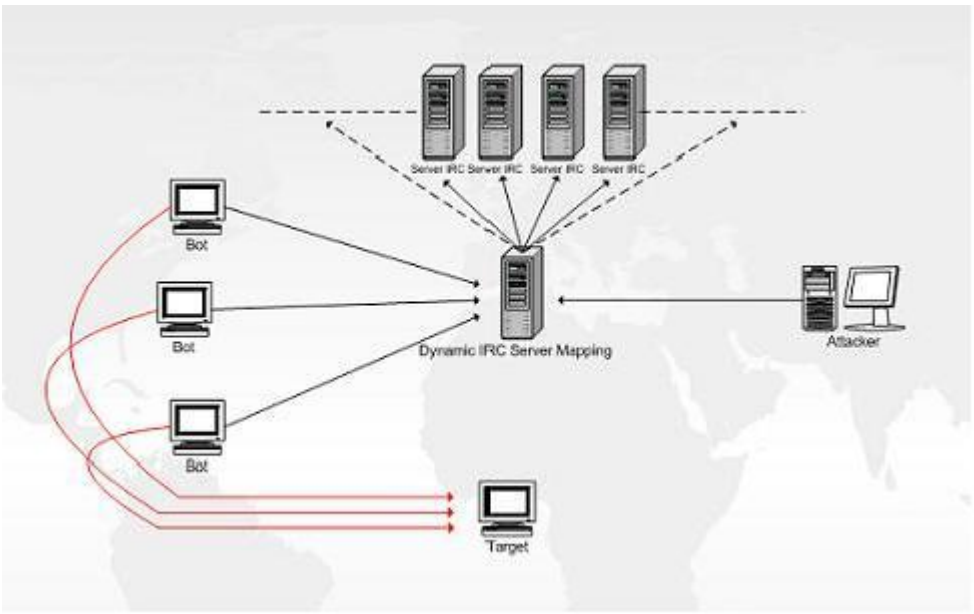


Figure 2.1: IRC botnet with multiple servers [9]

to take down the C&C, the DNS (Domain Name Server) records are targeted instead of IP addresses which is more difficult, but not impossible.

While IRC botnets are not immune to discovery and potential destruction, they are still the most common mainly because they are quite easy to set up and maintain.

2.3 Web-based botnets

There are botnets that use HTTP, FTP and other protocols for C&C. A botmaster sets up a web server to which clients can connect to and either open up a backdoor for communication with the C&C or download a file with instructions. This set-up is quite similar to IRC.

2.4 P2P botnets

P2P protocols became more common with Napster, the concept of nodes in a network acting as both clients and servers. Other protocols followed such as Gnutella, eDonkey, BitTorrent and Kademlia among others. These protocols are completely decentralised which is attractive for botmasters. Peer-to-Peer (P2P) botnets are so named because they use P2P technologies and protocols either for spreading, for C&C or both. Some of these use a centralised structure

like IRC botnets while others use a distributed structure as can be seen below 2.2.

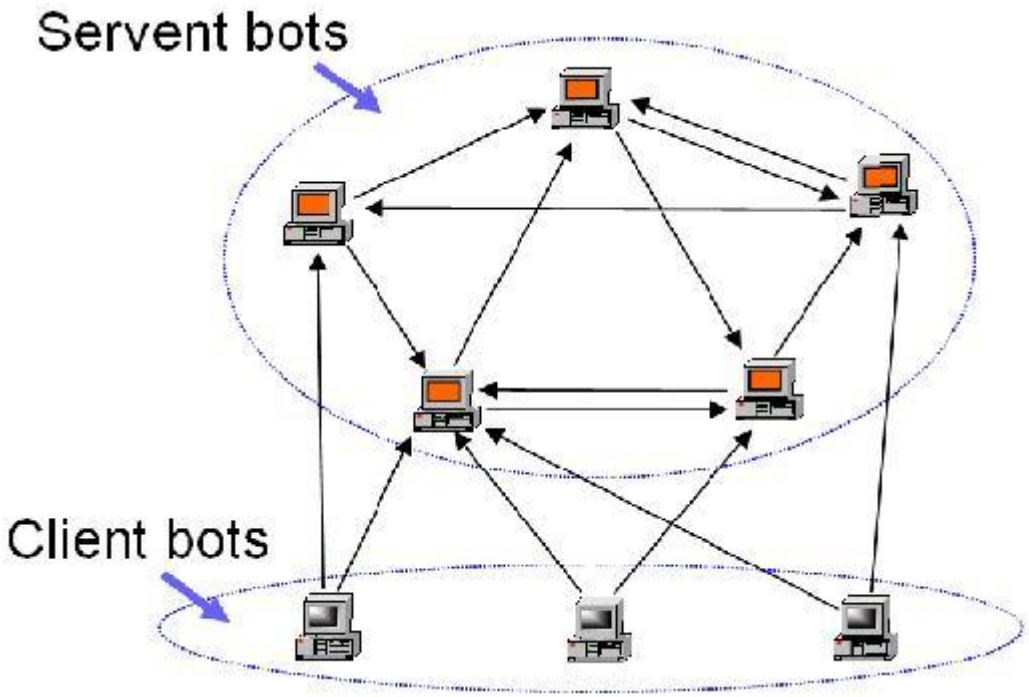


Figure 2.2: p2p hybrid botnet

[11]

There are clients and servents (computers that are both client and server) in such an architecture. The servents are chosen because they have static IP addresses, to avoid having to change peer list addresses too frequently.

C&C: Each servent keeps a list of peers in its group and when it receives a new message from the botmaster, it automatically sends it to the list. For control the botmaster can use any servent to send commands from. If a group of servents are compromised, this causes no problem to the rest of the botnet. Certain servents act as sensors and they dynamically change. Communication is encrypted and commands are authenticated using Public Key Cryptography. [12]

Most of the P2P botnets that encrypt their communications use asymmetric encryption for C&C and symmetric key encryption for client to client communication. This makes it harder to detect malicious code (the IDS has to be able to read it to recognise it as malicious). A red flag may be raised if encrypted traffic is found in a stream that should not be encrypted, so some botnets are smart enough to use the same ports that other encrypted services like SSH (Secure Shell) or HTTPS (Secure HTTP) use. Some of them can even mimic them to reduce suspicion.

Some P2P botnets and their vulnerabilities from [12]: Most of the vulnerabilities lie in peer discovery. While P2P botnets can be prevented from growing, there is no method of mapping the whole botnet and possibly shutting it all down for the moment.

Nugache: It has an initial constant list of 22 peer IP addresses hardcoded into the bot program. If these 22 are compromised, the botnet stops growing because new bots have nowhere to connect to.

Phatbot: peer discovery is done by finding Gnutella clients with ID (Identity) "GNUT" listening on unusual ports. With the cooperation of the server admins, these ID's can be shut down and prevent botnet from growing.

Sinit: this one uses P2P technology just for peer discovery. It has no list to begin with, instead it randomly probes different IP addresses and since there are over 4 billion of them, this is not the most effective method. It always communicates on port 53 and all the packets have the same format which makes it easy to detect. Again only growth can be limited.

Spamthru: this is a true P2P and central C&C hybrid. It first uses the centralised architecture, but if a C&C is taken down, it uses P2P to choose another C&C. The weakness is that instead of shutting down the C&C one can use it to map out the whole botnet.

I have decided to focus my efforts on a new botnet known as Storm Worm (or Peacomm worm, it has many names) which is the main first full P2P botnet. It uses the Overnet protocol for both communication and control and propagation.

2.5 Storm Worm

This botnet (heretofore simply referred to as Storm) first emerged in early 2007 and spread by sending e-mails with currently relevant subjects such as natural disasters (storms mostly- hence the name) and other topics in the news with attached videos/images. It would also use holidays as a topic in its spam mails and would attract people to websites claiming to have greeting cards, pictures, videos and free games for download. For example in December, 2007 it started exploiting the Christmas season by sending Christmas themed emails leading victims to a website called "merrychristmasdude.com" where they are encouraged to download pictures. The videos and images are in fact executable files and have the extension .exe, but they count on people to not notice this.

To clear up any misunderstandings, while Storm may have started off as a worm, it no longer is. It spreads mainly using social engineering methods, like tricking people into donwloading it from e-mails or websites. It does not roam the Internet looking for vulnerabilities in machines that it can exploit. It requires the active participation of the victim to infect machines, so it's not automated. The problem is that the websites can be taken down. Storm has solved this by using fast-flux service networks. The website's DNS records change every few minutes. This means that each time a client connects to the website, he/she is actually connecting to a different IP address. Fast-flux service networks are explained further in its own section below. The solution is to blacklist these websites as they're discovered.

2.5.1 Infection process

After the executable has been downloaded and executed, it adds the system driver "wincom32.sys" to the Windows process "services.exe". Part of the installation involves hardcoding a peer list on the bot and saving it in a file called windir/system32/wincom32.ini. The Windows firewall is then disabled and several TCP (Transmission Control Protocol) and UDP (User Datagram Protocol) ports (list of ports to be added) are opened. The worm then bootstraps the bot onto the Overnet network so that it can contact its peer list if they are online.

```
[peers]
1: <128 bit md4 hash>=<IP address><Port><2 byte flag>
2: <128 bit md4 hash>=<IP address><Port><2 byte flag>
   ...
N: <128 bit md4 hash>=<IP address><Port><2 byte flag>
```

Figure 2.3: Contents of wincom32.ini

[2]

The peer list 2.3 contains a number of addresses in this format. The list can be used to take down part of the botnet or to monitor the addresses provided and help estimate the size of the botnet. This is of course in assuming that the botnet is not organised into cliques or cells.

After connecting to the Overnet network and contacting its peer list, the bot is then ready for the secondary injections. It searches the Overnet for a certain value (the search key is hardcoded into the bot) and downloads it. This value is an encrypted URL (Uniform Resource Locator) that points to a second executable. After the bot has downloaded and executed it, it can then get on with

business. There are several injections, some of the secondary injections lead to others or program the bot to periodically automatically check for updates on the network. The injections a bot has to go through are:

1. Downloader and rootkit component
2. SMTP (Simple Mail Transfer Protocol) email spamming component
3. Email address harvester for the previous spamming stage
4. Email propagation component
5. Distributed denial of service tool

2.5.2 Properties

The main properties of Storm are: It has aggressive defences (it attacks anyone who tries to analyse it or reverse engineer it). It uses a clique architecture where each clique has its own 40-bit encryption key. There are no file exchanges between infected hosts which makes it difficult to track.

Storm Worm is now employing the same tactics as terrorist organisations. Imagining a botnet as a large network where every node knows all the other nodes, if one is captured, this will reveal the identity of all the other nodes and they can in turn also be taken down. This would be disastrous. The other option is to have cells, where each node belongs to a cell and knows only the members of the cell. If a cell is taken down, it is does not put the whole botnet at risk. The botmaster can hide better because he could be a part of any cell and can in fact move around to different cells.

When the botmaster needs to relay information to all the nodes, he notifies one member of each cell who then passes on the information. This creates less traffic than if he were broadcasting to all the bots. Another advantage is that even if traffic is being monitored on a certain bot, suspicion is not raised if it keeps receiving messages from the same source. The fact that the C&C server keeps changing also confuses matters even more. At the moment of writing, there is no practical way to track a Storm botnet to its master.

Some articles report possible ties to RBN (Russian Business Network), a Russian ISP (Internet Service Provider) that is believed to be the source of several malicious softwares. [13] This is mere speculation, but Russia is well known for having very lax laws when it comes to cybercrime. [3] [14] Many criminals who want a “safe” ISP base their operations in Russia, although they may not

be there physically. One of the main problems with cyber crime is that the Internet does not follow the physical geography. This makes it hard to locate and capture the criminals.

2.6 Fast-Flux

Fast-flux is a technique that changes public DNS records every 3-10 minutes. This is accomplished with a network of infected computers (very much like a botnet). The main idea is to allow a domain name to map to hundreds or thousands of IP addresses. This makes it extremely hard to take down a website that has criminal content. [14]

Fast flux service networks behave like botnets in more ways than just a network of zombies. They also regularly contact the motherships to receive updates and other command or management operations. Most C&C observed uses HTTP as its base protocol.[14]

While changing IP addresses every few minutes makes it hard enough to take down a website, there is an additional layer of security: blind proxy redirection. This means that the IP addresses used are for a front of proxy machines that redirects clients to the real servers. These background servers act as the "motherships" and can be compared to the C&C in a botnet. Any directional protocol with a specific target port can be used in fast-flux service networks. Protocols like SMTP, IMAP (Internet Message Access Protocol), POP (Post Office Protocol), HTTP have been used in fast-flux networks.

There are two types of fast-flux: single flux and double flux.

Single flux: Single flux networks comprise of compromised PCs. The client does a DNS lookup on a URL and gets an IP address in return. This is how DNS lookups normally work, except that in this case, the IP address will change every few minutes. This IP address can either lead to one of the PCs in the flux network or to a *bullet-proof* DNS server. A *bullet-proof* DNS server is a hosting service in countries with lax cybercrime laws such as Russia, China and other countries.

Double flux: This technique adds another layer of obfuscation to the single flux. Both the DNS A records and the authoritative NS records keep changing. A client does a regular DNS lookup, and in return gets an IP address of a PC in the flux network. However this PC is merely a proxy and it contacts the mothership to get the actual content and returns it to the client.

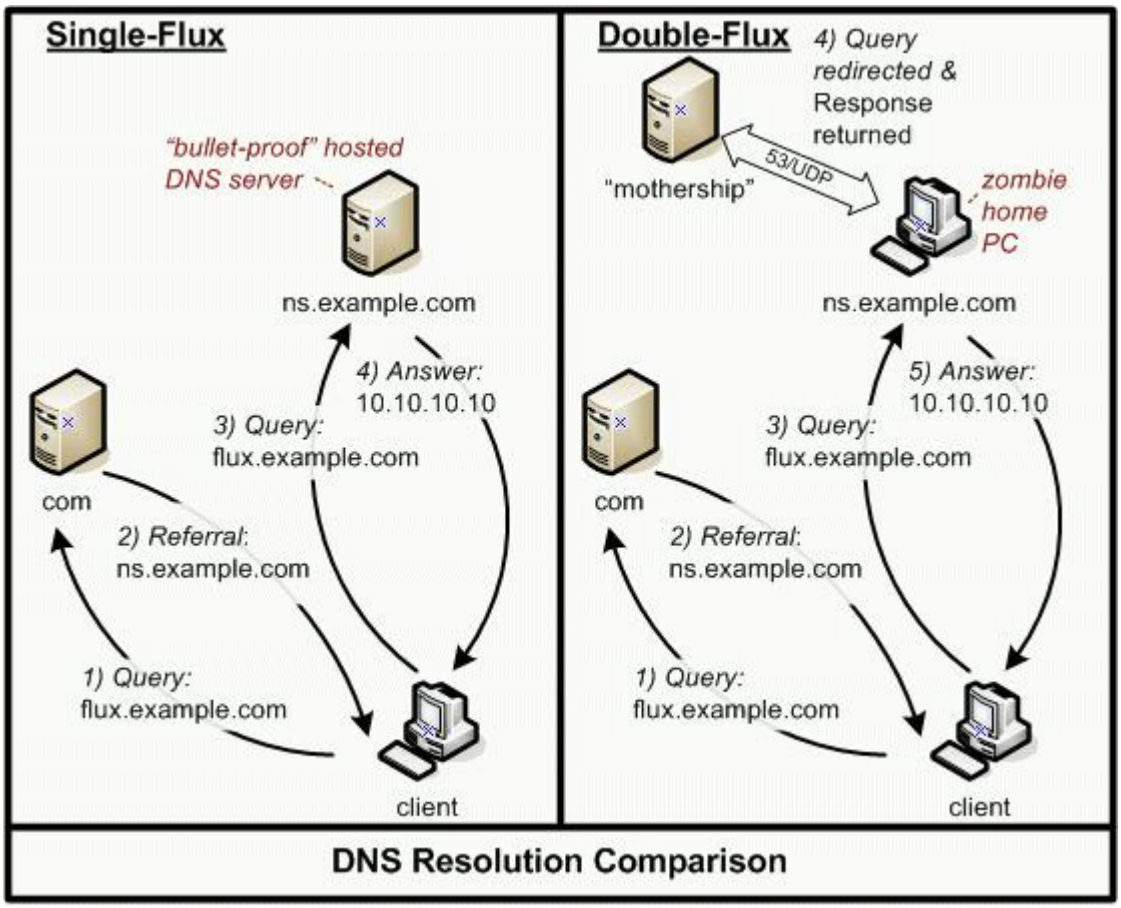


Figure 2.4: Single-flux and Double-flux
[14]

The reasons fast-flux works so well as a criminal tool are:

1.They're simple to set up, just like IRC botnets. One powerful mothership to act as a DNS and C&C server is all that's needed. For operations, there are several botnets that can be rented or bought.

2.There are small chances of getting caught. Even if the front proxies are discovered, it does not matter if they're disconnected, there are many more to take their place. Tracing any C&C is impossible since the traffic logging on the victim computers is disabled.

3.Due to "criminal-friendly" hosting services, shutting down the servers can be extremely difficult. Even if the authorities manage to work around the fast-flux technology and discover these servers, they need the cooperation of the ISPs.

2.7 Overnet

Overnet is a peer-to-peer network based on the algorithm Kademlia. Overnet was joined with the file sharing application eDonkey2000. They were both taken down in late 2006 after legal action from the RIAA (Recording Industry Association of America) for illegal file sharing.

The Overnet network however is still in operation because it has no centralised control that can be shut down. This is what Storm uses for communication and peer coordination. The Overnet protocol is closed-source and there is little literature on it. As the official Overnet website (www.overnet.com) has also been shut down, it's even more difficult to find any information on its structure.

Kademlia is the algorithm used as Overnet's Distributed Hash Table (DHT) protocol. "Kademlia assigns a 160-bit hash ID to each participating peer, and computes an equal-length hash key for each data object based on the SHA-1 (Secure Hash Algorithm) hash of the content." [15]

2.8 Forensics basics

According to the McGraw-Hill Dictionary of Scientific and Technical Terms [16], digital forensics is "The study of evidence from attacks on computer systems in order to learn what has occurred, how to prevent it from recurring, and the extent of the damage.".

There are different ways of performing digital forensics, but the basic principles of forensics still apply. Here are the "rules" as summarised by [17]:

Rule 1. An examination should never be performed on the original media.

Rule 2. A copy is made onto forensically sterile media. New media should always be used if available.

Rule 3. The copy of the evidence must be an exact, bit-by-bit copy. (Sometimes referred to as a bit-stream copy).

Rule 4. The computer and the data on it must be protected during the acquisition of the media to ensure that the data is not modified. (Use a write blocking device when possible)

Rule 5. The examination must be conducted in such a way as to prevent any modification of the evidence.

Rule 6. The chain of the custody of all evidence must be clearly maintained to provide an audit log of whom might have accessed the evidence and at what time.

When conducting a forensics analysis on a computer, there are some important things to think about. *The order of volatility*. The system keeps changing and the data gets altered continuously on a computer, therefore it's important to capture the data before it is modified or lost. The order of analysis should reflect this, with the more volatile data collected first. *Offline vs. Online analysis*. The best method is to perform an offline analysis, this means that there will be no changes to the system while the analysis is ongoing. An online analysis is however sometimes necessary, in the case of certain data being lost if the system is taken offline.

While gathering digital evidence is not an easy task, one must consider that most activity in computers revolves around the same files, programs and other resources. It is estimated that approximately 90% of files are unused for over a year.[18] Any unusual activity, such as the sudden frequent use of a previously underused file, will stand out. This is what one looks for while gathering evidence: unusual behaviour of any kind.

To get an accurate idea of what happened, one must look at the data on different levels. At the User Interface level, there is little useful information. The data is manipulated to present a human-friendly image. This is not the real data. Going too deep to the bit level is also not advisable, as all the bits look the same, so there is not much to learn at this point. Somewhere in between is where the real information lies. [18]

There are many forensic tools, commercial and open-source that can be used on different systems.

Chapter 3

Literature Review

3.1 SRI International

The SRI published a static analysis of Storm worm and specifications of the changes made to their bot detection system BotHunter to include Storm worm detection. Their analysis is based on the variant released on 2.September, 2007 named "labor.exe". This version of Storm as is the case with other newer versions does not check for virtual environments.[10]

The downloaded binary is encrypted with a 40-bit symmetric key. After decryption, the code begins to execute and it first produces a copy of its original encrypted form called spooldr.exe. It then modifies the tcpip.sys driver and creates a driver called spooldr.sys. The modification to the tcpip.sys driver makes it so that whenever the infected machine is rebooted, the driver will automatically spawn spooldr.sys. It then creates a list of processes and drivers that should be monitored and immediately terminated if they start to run. Another job is to clean up any old storm worm code that may be on the machine and other malware that would interfere with its running.

Below 3.1 is an overview of Storm's logic:

The initialisation phase starts with the creation and running of the file *spooldr.ini*. Security is implemented to protect the process from probing during execution. The network is initialised by implementing the appropriate Windows Sockets version.

The next phase is to connect to other infected peers over the Overnet network. There are 290 peers on the list in spooldr.ini and they're all contacted within 30 seconds. This is repeated every 10 minutes if there is no response. Once a successful connection has been made, updates are downloaded and applied and attacks (if a command has been given) are carried out.

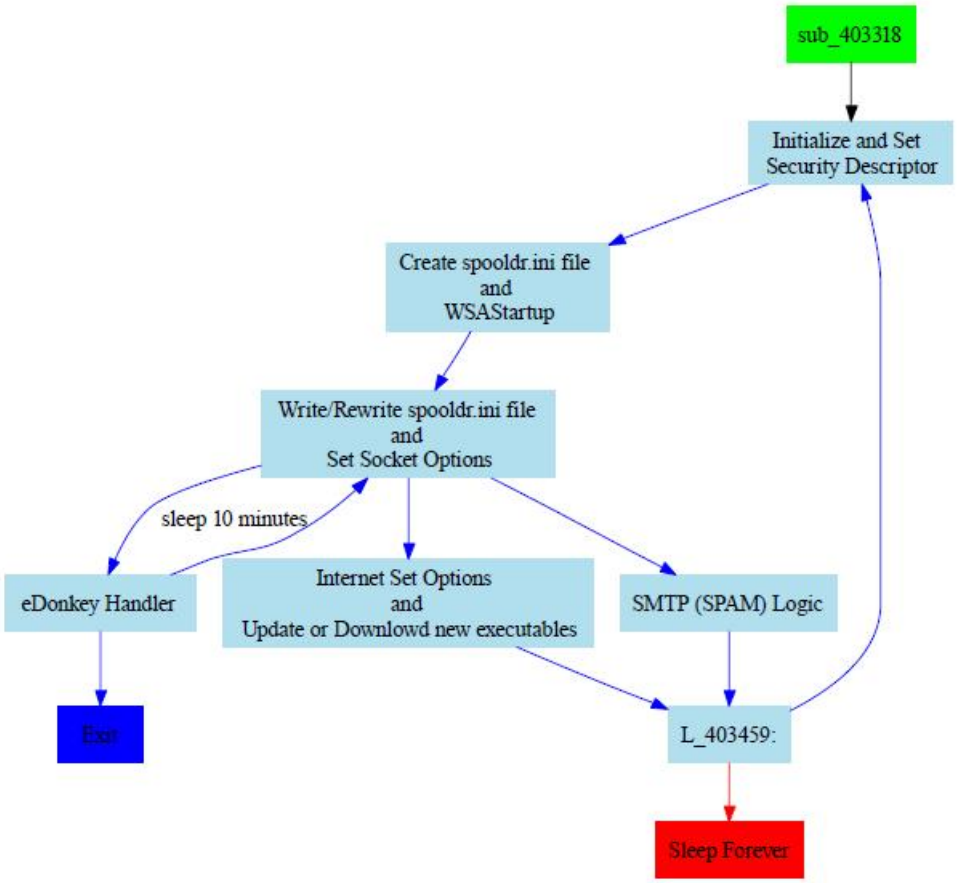


Figure 3.1: Storm Logic Overview

[10]

Storm uses high-order UDP ports for its Overnet communications. The packets sent are characterised by the beginning sequence (0xe3). The next byte indicates the type of message. From observing these messages, certain dialog sequences were deduced. [10]

The sequence for Overnet communication with other peers to either publicise themselves or update peer lists is called the "Overnet Control Plane Dialog" 3.2. As can be seen from the figure below, there are 4 different types of messages. They're not always in that sequence, it can be just the first two or the last two depending on the type of communication going on. The "publicize" and "publicize ACK" are for when an infected machine wants to inform others that it has joined the group, it then receives acknowledgements from the other peers. The "connect" and "connect reply" messages are for connecting to other peers for updates or just to check that they're there (like in the initialisation phase).

Another sequence is the "Overnet Data Retrieval Dialog" 3.3. In the first message, a peer A issues a search on a particular hash. The "Search Next" message

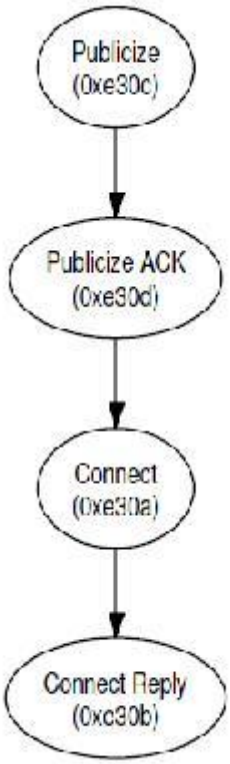


Figure 3.2: Storm Overnet Control Plane Dialog flow
[10]

is sent by the other peers in response and contains suggestions of which peers have what A is looking for. Peer A then sends a "Search Info" message to restrict the search. The peers in question reply with a "Search Result" message that contains filecontent hash and meta data for all the files that correspond to the searched hash. They then send a "Search End" to indicate that there are no other results available. Peer A selects which file is appropriate and launches a new search dialog with the filecontent hash as the search key. The dialog sequence continues until there is a result with location information, normally an IP address and a port number.

The "Overnet Data Publishing Dialog" 3.4 is used when a peer wants to publish a file. MD4 (Message Digest 4) hashes of all the file's keywords are computed and published along with the metadata of the filecontent. Then the filecontent hash is used as the key to the location and the hash of the file's owner.

Deviations from normal dialog flow: There are some exceptional behaviours that don't have a bearing on Storm. There's speculation that Storm clients distinguish themselves from other eDonkey clients by not responding to certain eDonkey messages. This is not proven in [10] and has not been observed by any of the others. If it were possible to distinguish Storm clients so easily, they

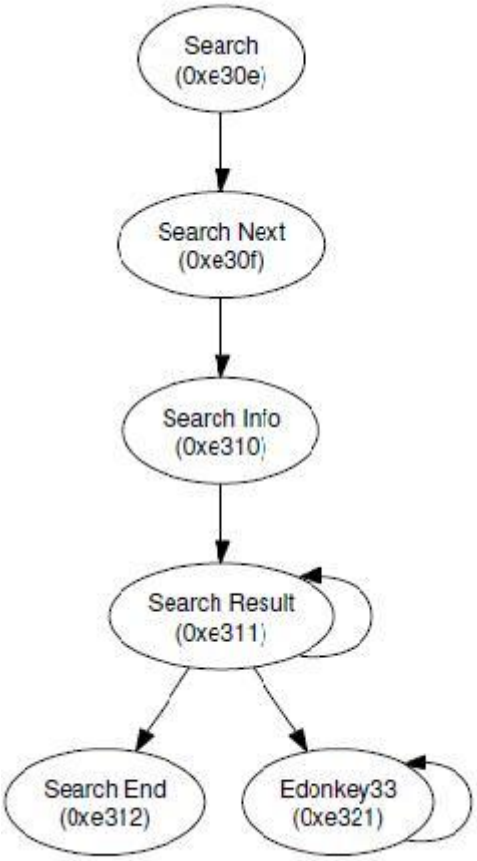


Figure 3.3: Storm Data Retrieval Dialog Flow
[10]

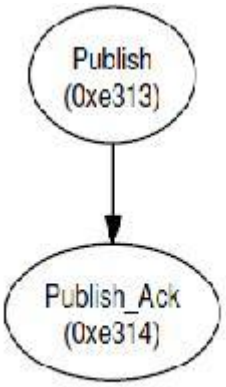


Figure 3.4: Storm Overnet Data Publishing Dialog Flow
[10]

would be much easier to detect than they have so far proved to be. Traffic analysis on infected hosts shows that apart from the first 5 minutes, almost all TCP traffic is primarily spam. There's a continuous stream of spam emanating from the machines averaging 20 per minute.

3.1.1 BotHunter

BotHunter is a tool developped by SRI to detect botnet infections. It is based on Snort and uses Snort results as input. BotHunter uses what is called "dialog-based correlation". This means that it looks at certain sequences of events; dialog between the machine and the outside to detect botnet activity. They have modelled the "normal" dialog sequence of different types of botnets and Snort results are used to match the sequences to the stored ones. For a detailed specification of BotHunter, see [19]. Here I will only focus on its ability to detect Storm.

The Storm dialog state model in BotHunter is shown below 3.5:

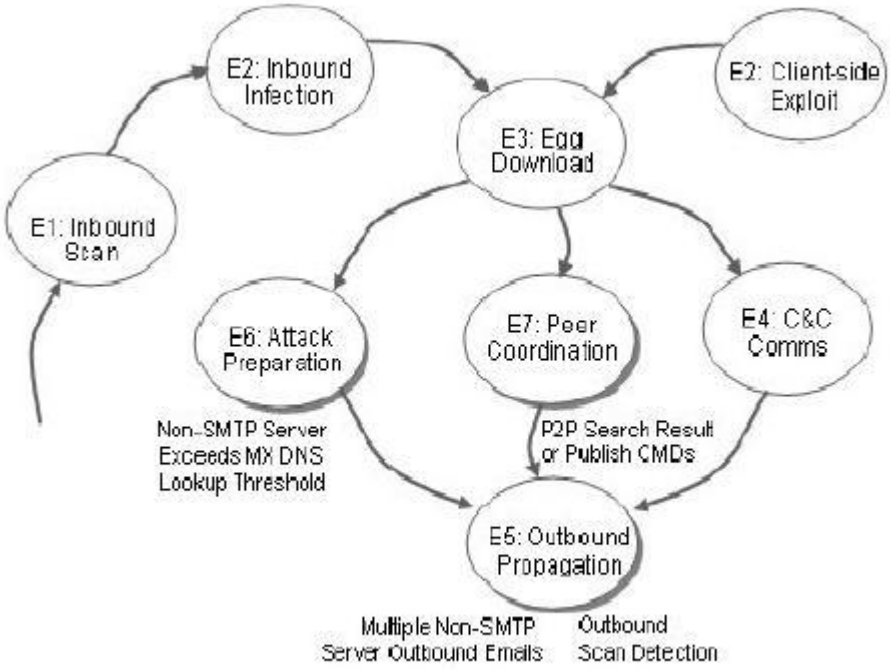


Figure 3.5: Storm Dialog State Model

[10]

The states:

1. Inbound scan: This is the scanning process that worms go through to select victims. Storm does not do this anymore, as explained earlier it relies more on social engineering to infect hosts.
2. Inbound infection: usually downloaded from a URL.
3. Egg download: This stage comes later with Storm, depending on how long it takes to connect to the Overnet network, connect to other peers and download a secondary injection.

4. C &C: This is not applicable to Storm since it doesn't use any central C&C server.
5. Outbound propagation: This is characterised by a considerable increase in SMTP and TCP and UDP outbound traffic.
6. Attack preparation: This is a stage when the host starts preparing for attacks, such as collecting mail host IP addresses.
7. Peer coordination: in this state, BotHunter looks for Overnet protocol communications, especially the ones with Search and Publish message sequences.

BotHunter does not need to detect all these sequences in the correct order to issue an alert. With Storm, it's enough with 2 of the outbound states (5-7) in any order. More on BotHunter results in the honeypot chapter.

3.2 Joe Stewart for SecureWorks

This report was published in February, 2007. It was prompted by DDoS attacks on some anti-spam websites and the report focuses on the DDoS aspect of Storm. The version of Storm under study in this report has six different executable payloads downloaded at different stages of infection that serve different purposes.

- game0.exe - Backdoor/downloader
- game1.exe - SMTP relay
- game2.exe - Email address stealer
- game3.exe - Email virus spreader
- game4.exe - DDoS attack tool
- game5.exe - Updated copy of Storm Worm dropper

[20]

The hardcoded peer list contains over 100 addresses in the form *hash=IP:port* in hexadecimal. According to [20]: "The hash value doesn't actually correspond to a file, it is generated using an algorithm which takes as input the current system time and a random number between 0 and 31, outputting one of 32 possible hashes for any given day." This hash value acts as an authentication token and a search for it returns another hash value to use as a key in conjunction with another hardcoded key. In addition to the decryption key hash,

the response packet contains a meta-tag "id". In its body there's a string value which is an encrypted URL that points to the secondary injection. *DDoS attack:*

game4.exe is hardcoded with the target IP address and the attack type (a port 80 TCP syn flood or an ICMP (Internet Control Message Protocol) ping flood, or both). Some targets were profit motivated, perhaps an assignment from a third party. Other targets were anti-spam websites and others that tried to analyse or reverse engineer Storm. There was even an attack launched against a rival spam group. This agrees with the opinion that Storm is highly aggressive.

3.3 Grizzard

This is one of the earlier analyses of Storm, refered to here as Trojan.Peacomm. They used a honeypot set up to observe the trojan's behaviour. For analysis of the binary, PerilEyez [2] was used.

Observations: The binary executed and added the system driver "wincom32.sys" to the Windows process "services.exe". This service was subsequently used as a client to connect to the Overnet network.

The ICF/ICS (Internet Connection Firewall/Internet Connection Sharing) was set to "disabled". TCP ports 139 and 12474 and UDP ports 123, 137, 1034, 1035, 7871, 8705, 19013 and 40519 were opened. The peer list contained 146 peers and was hardcoded into the host.

Storm's communications behaviour after initial infection was categorised into the following steps [2]:

1. Connect to Overnet: publishing and connecting to peers.
2. Download secondary injection URL: searches for hard coded hash keys on Overnet and downloads the encrypted URL that is location for the secondary injection.
3. Decrypt URL: the decryption key is hardcoded.
4. Download secondary injection.
5. Execute secondary injection.

These steps are fairly similar to those observed in other papers. They discuss that the initial peer list can be a weakness. If they were to be shut down, this would stop the bootstrap process of all the infected hosts with the same list. Another wekness is that they can be monitored to detect other infected hosts.

They don't mention how it would be possible to shut down 146 machines. As for monitoring, it's possible to do so, but the resources required would be enormous especially since the peers keep changing, and the lists keep getting updated as new connections are added and so on. It's also not very likely that the 146 peers would all be connected, so it would take a lot of time to monitor all of them.

Secondary injections include components for spamming, e-mail address harvesting, propagation via e-mail, DDoS tool and the rootkit. They were downloaded via HTTP from URLs in the form of http://xxx.xxx.xxx.xxx/aff/dir where xxx.xxx.xxx.xxx is an IP address. The payloads were observed to change after a while meaning that the binaries are updated after a certain time.

An interesting observation was the network trace they did to determine the traffic pattern of an infected host. They found that at first the network traffic is pretty low and immediately after the bot has been executed, the traffic shows a significant spike. The traffic curve keeps growing pretty quickly as the infected host tris to connect to more and more peers. Eventually it runs out of peers to contact and the curve starts levelling out. It keeps increasing over time as more peers join, but does not grow as quickly as in the beginning.

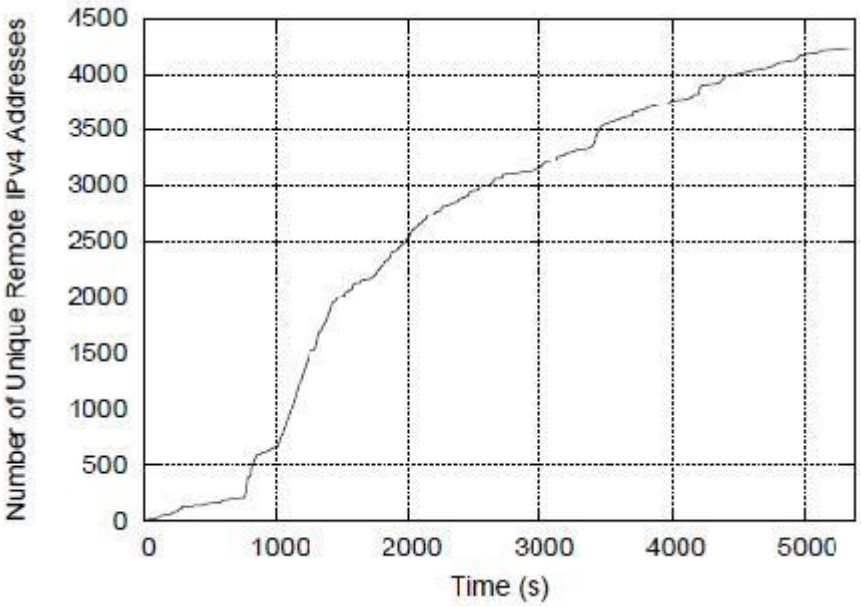


Figure 3.6: Unique IP addresses contacted over time

[2]

3.3.1 The search keys

They wrote a tool to analyse the search messages. This instance of Storm searched for five hash keys, of which the first one is its own IP address. This is part of the Overnet protocol to keep each host updated about its neighbouring nodes. Of the remaining four hashes, only two were found. There were five responses and four unique hosts responded. It took 6 seconds and 3 seconds for each of the hashes to be found. They speculate that this means that the command latency is higher than for centralised botnets. This seems to be a weak assumption if it's only based on observing two results. They do not provide further proof.

3.3.2 Identifying infected peers

Out of the 10 105 that were connected to the Overnet network at the time, their honeypot only communicated with 4200 hosts. It was difficult to know whch of the hosts are infected. They also were not sure that the four hosts who responded to the hash search were infected. At a later point, their honeypot receives a search request that it had also searched for earlier and they concluded that this host must also be infected with the trojan. It's not clarified why they would assume that a search request for the same hash is proof of infection and yet responding to the search is not.

Chapter 4

Security measures against Storm Worm

4.1 Prevention methods

In the case of Storm worm, the best prevention is knowledge. Their primary infection method is to trick people into downloading harmful content. This indicates that they're counting on people to not know any better and unfortunately they're right. People continue to download attachments in e-mails from people they don't know, they download videos and pictures from unknown websites and have the tendency to assume that just because they have an antivirus program they're safe.

Many corporations also don't take responsibility for educating their employees because it's technical knowledge is relegated to the IT (Information Technology) group. This is a grave error since no matter how good the IT support group is or how good the IDS/IPS is, people are still the main cause of security issues. According to surveys done by CompTIA, human error is the leading cause of security breaches in the polled companies.[21] While this trend is diminishing, it's still a major problem especially since the poll also shows that while security breaches are fewer, they have greater consequences now. On their private computers, people are even less careful than on company computers. They don't have the restrictions of a security policy and a team of experts.

While a company can provide training to its employees, there's very little to do in the case of individuals in their own homes. Many homes now boast a broadband Internet connection and many people leave their computers on while connected to the Internet even when they're not using them. These computers

are usually running some version of Windows and there are no sophisticated IDS protecting them. In this case, not much can be done really. Educating everyone on the dangers of malware over the Internet would be a daunting task and I have no recommendations on how to solve this problem. Perhaps government sponsored public service announcements might help.

In an organisation, the best prevention is to block P2P communications, which many companies do. They can block out specific protocols on their IDS to prevent Overnet communication. This would help because even if the computers are infected, they would not be able to coordinate with other peers or to download secondary injections.

4.2 Detection methods

In the case of an organisation network, the NIDS/NIPS (Network IDS/IPS) can be configured to warn against any Overnet protocol traffic. General rules forbidding P2P traffic would be a good idea. People do not always follow the rules however, so it's a good idea to have these protocols raise alarms in the IDS. Monitoring traffic to make sure there are no PCs with increased outgoing traffic is also a good way to detect a possible Storm infection.

Sebek [22] is a very good tool for detecting any sort of unusual behaviour, even the ones that are encrypted or otherwise obfuscated. It is particularly useful in the case of inside attacks. These bypass the NIDS. There can be Sebek clients installed on the machines and Sebek servers on the network management machines. This is additional information to store and sift through, but there are considerable advantages:

1. Sebek detects inside attacks. A considerable amount of attacks come from inside the organisation, malicious or accidental.
2. Certain malware may bypass the NIDS, for example, by using encryption. These would still be caught by Sebek.
3. Physical attacks on the machines. Such as someone attempting to physically break onto a machine. Another advantage is that Sebek is generally undetectable and the machine users would not be aware of its presence.

4.3 Defensive Measures

Companies should have a good IDS that would alert them to infections. Storm Worm is notoriously difficult to detect, but there are many researchers working on the problem. Keeping up with the latest research on the topic is also a good idea because the Storm worm writers are also keeping up. Some researchers have come up with practical tools for detection such as BotHunter from SRI International. These tools need to be tested in a "real" setting and it does no harm to install them.

Chapter 5

Honeynet Experiment

5.1 Introduction

The idea is to set up a honeynet comprising of a machine running VMware [23] with virtual Windows XP and a real Windows XP machine. The two machines are both connected to the Internet, but all the traffic from and to them goes through the Honeywall. They are running Sebek clients that collect data from the machines and send it to the Sebek server on the honeywall.

The reason for running two machines, one virtual and one real, is because some versions of Storm can detect a virtual environment. In the case that the virtual environment is detected, I'll still have the set of data from the real machine. If it's not detected, then it would be interesting to see if there are any differences in Storm's behaviour and effects on the systems.

The honeywall has snort-inline as IDS and runs the Sebek server that stores the information from the two Sebek clients.

The purpose is to infect the two honeypots and then collect data using the honeywall. The WallEye web application is used for network data analysis, while [forensic tool of choice] is used to analyse the honeypots' system data.

5.2 Set-up

The honeynet is set up as follows 5.1:

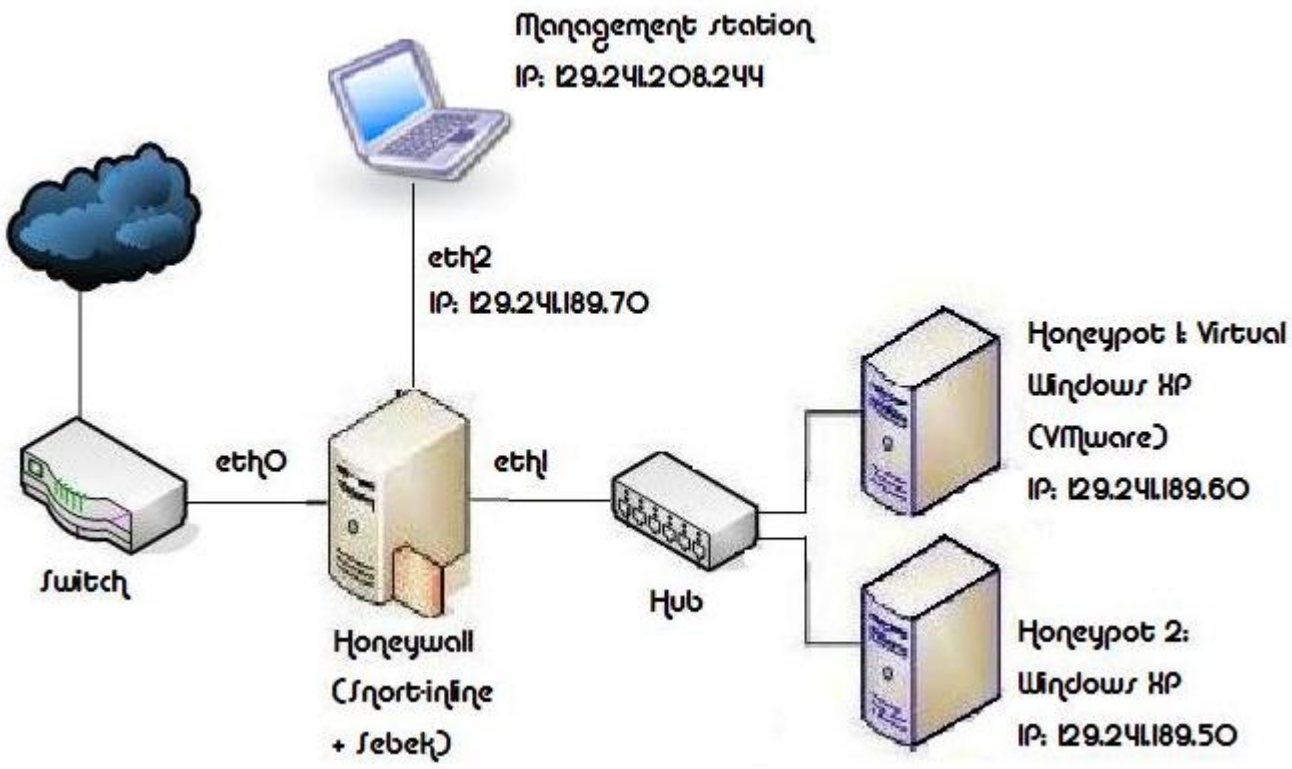


Figure 5.1: The Honeynet

5.2.1 Honeypot 1

This is a Linux Ubuntu machine. I first installed VMware on it and created a Windows XP Professional virtual machine. This machine is directly connected to the network and has a private IP address. I then installed the Windows Sebek Client version 3.0.4 on it and configured it to send all its data to the honeywall machine. As explained in the Sebek section, this client will be completely invisible to any user of the machine. The data will be sent via a covert channel and will therefore not be detected by any traffic sniffers.

Host: Linux Ubuntu Virtual Machine (VM): Windows XP VM IP address: 129.241.189.60

5.2.2 Honeypot 2

This is a Windows XP machine at a bare minimum. There are no additional applications as they are not necessary. Only a Sebek client was installed and configured exactly as on Honeypot 1. The consequences of infecting this machine directly are that it will have to be formatted after the experiment. On the other hand, it provides a good back-up plan in the case the virtual machine is detected. The security on this machine is very limited, although the Windows

Firewall is turned on. Storm is supposed to turn it off.

Host: Windows XP IP address: 129.241.189.50

5.2.3 Honeywall

The honeywall used can be found at [24]. It captures all the data that goes in and out of the honeynet, it can also control the data to ensure that no harmful activity goes beyond the boundaries of the honeynet. It also has data analysis capabilities through WallEye. The important thing is that it has to be between the Internet and the honeynet and it must be invisible to attackers, meaning that it has no IP address on neither the internal nor external interfaces.

Another vital use of the honeywall is to prevent our honeynet from being used to attack other systems, such as being used in DoS attacks or to send malicious code to other hosts. DoS attacks are prevented by limiting the amount of traffic going in or out of the honeynet. This however does not prevent infecting other hosts, for this snort-inline is used as a reverse intrusion detection system, meaning that it prevents malicious code from exiting the honeynet. It examines all outgoing IP traffic and either drops the packet or modifies it.

In this set-up the honeywall is configured using the dialog menu. Some of the fields were left with the default values, but the IP addresses of the honeypots and the management interface had to be added. Remote access to the machine also was configured to only allow one machine. The traffic limits also had to be set to prevent any DdoS or spam attacks emanating from the honeynet. The Sebek server just takes any Sebek packets it receives on the interface, so there was no need to configure the Sebek server for reception of packets.

Some important configuration values are:

The honeywall is bridged and therefore has no IP address. It's not supposed to be visible to the outside world.

The internal interface is eth1.

The external interface is eth0.

The management interface is eth2 and has IP address 129.241.189.70. This address is important because it's used for remote logon on the web interface.

For the complete configuration file, please see Appendix A.

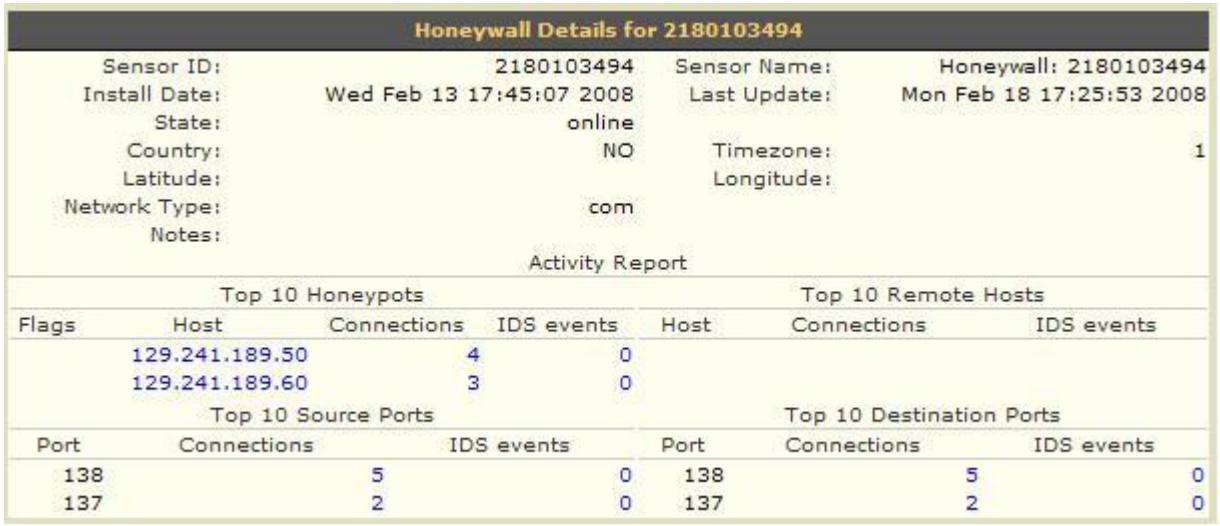
5.2.4 WallEye

This is the web interface for management and data analysis of the honeywall. To connect to the honeywall, the SSHD (Secure Shell Daemon) port on the honeywall must be open. There are only two TCP ports on the honeywall management interface that accept incoming traffic: 22 and 443. 22 is for the SSHD connection, 443 is for the management connections.

The management interface can only be accessed from a pre-configured IP address, in this case, my personal workstation (129.241.208.244) To connect to Walleye, the url: https://management-interface-IP-address. Note that SSL is used. After the certificate has been accepted, there's a login screen where the password is changed to a more secure one.

The main view is of the general overview of the honeywall traffic in general. There are several management options, such as viewing the traffic on a specific IP address or viewing a specific type of traffic. The different options will be explored in the results section.

Below is a screen capture that shows both honeypots and a summary of the traffic going in and out of them.



Honeywall Details for 2180103494

Sensor ID:	2180103494	Sensor Name:	Honeywall: 2180103494
Install Date:	Wed Feb 13 17:45:07 2008	Last Update:	Mon Feb 18 17:25:53 2008
State:	online		
Country:	NO	Timezone:	1
Latitude:		Longitude:	
Network Type:	com		
Notes:			

Activity Report

Top 10 Honeypots

Flags	Host	Connections	IDS events
	129.241.189.50	4	0
	129.241.189.60	3	0

Top 10 Remote Hosts

Host	Connections	IDS events

Top 10 Source Ports

Port	Connections	IDS events
138	5	0
137	2	0

Top 10 Destination Ports

Port	Connections	IDS events
138	5	0
137	2	0

Figure 5.2: Honeynet Details

5.2.5 Other equipment

- 1 Hub
- 1 Switch
- 1 LCD screen

- 1 KVM Switch (4 inputs Keyboard, Monitor and Mouse switch)

5.2.6 Problems encountered

The set-up was not straightforward. One of the problems encountered was that the machines had been used for previous honeynet set-ups and they had not been formatted afterwards. I eventually realised that it's easier to reinstall all the systems and start over from scratch. It's hard to know which parts of a previously installed system are operational.

One of the computers could not handle VMware and had to be replaced. The screen also had to be replaced because the old one could not handle the resolution required by Ubuntu.

After connecting the machines to the Internet and assigning IP addresses, I found out that I could not contact them from the outside (using ping) even though they could connect to the Internet without a problem. This was caused by the Windows Firewall blocking all inbound ICMP traffic. This has to be changed in the firewall settings.

The firewall was surprisingly not very difficult to set up. This is mainly because the configuration options are quite well explained and many of the default options are already set. It helps to read the User Manual [25] before trying to configure it.

Another good idea is to look at previous projects' configurations [26] to get a better idea of which values are expected if there are any doubts . It was configured using the Interview option and when asked to specify TCP ports for the management interface, it's mentioned that the SSHD port (22) is automatically added. This is not the case. I had to add it myself to connect to it via the web application.

For any other problems with the machines, Pål Sturla Sæther and Asbjørn Karstensen are extremely helpful.

Sebek proved to be a problem. The latest Windows versions Sebek 3.0.3 and 3.0.4 did not work well with the Walleye application. The packets are collected, logged and used as input in snort-inline, but are not identified as Sebek packets by Walleye. I could not find the solution to this problem, although I found many references to it on various security forums on the Internet. Instead of trying to figure it out, due to time constraints, I chose to continue the experiment theoretically.

5.3 Storm Infection

After the successful set up, it was then time for infection. Since this thesis is merely concerned with Storm infections, the infection was more active than passive. Storm requires that the victim downloads it and executes it. I could not wait to receive a spam mail containing Storm, so I used Storm binaries obtained from [27] and infected the machines with them.

Results and data analysis are in the next chapter.

5.4 Software

The software tools used in the set-up are briefly introduced below.

5.4.1 Honeywall Roo CD Rom

The honeywall CDROM is downloaded from the Honeynet Project and Research Alliance. It provides data capture, control and analysis capabilities. Most importantly, it monitors all traffic that goes in and out of the honeynet. It can be configured to block unwanted traffic in either direction, but its main purpose is to prevent the honeynet from being used to attack other machines. It runs Snort-inline, an Intrusion Prevention System based on the Intrusion Detection System Snort.

Snort-inline either drops unwanted packets or modifies them to make them harmless. [28] It records information of all the activity in the honeynet using Sebek. It runs the Sebek server, while the Sebek clients run on the honeypots. The clients then send all captured information to the server. For management and data analysis, it uses the Walleye web interface. Walleye also works as a maintenance interface, but there's a command line tool and a dialog menu that can also be used to configure and maintain the honeywal. [25] It runs on a Fedora Core 3 platform, but the plan is to make it OS independent in the future.

5.4.2 Sebek

Sebek is a data capture tool which mainly records keystrokes, but also all other types of sys_read data. It records and copies all activity on the machine including changes to files, network communications, etc. The main method it uses is

to capture network traffic and reassemble the TCP flow. This is in the case of unencrypted data. Encrypted data is another problem, because Sebek can only reassemble it in its encrypted form. Instead of breaking the encryption, Sebek circumvents it by getting the data from the Operating System's kernel.

Sebek has a client-server architecture. On the client side, it resides entirely in the Operating System kernel. Whenever a system call is made, Sebek hijacks it by redirecting it to its own *read()* call. This way Sebek can capture the data prior to encryption and after decryption.

After capturing the data, the client sends it to the server, which saves it in a database or simply logs the records. The server is normally on the honeywall machine in the case of a honeynet, and it collects data from all the honeypots and puts it all together for analysis.

To prevent detection by intruders, Sebek employs some obfuscation methods. On the client, it's completely hidden from the user and therefore from an intruder on the system as well. This is however not enough because the data captured has to be sent to the Server, thereby exposing itself. Sebek uses a covert channel to communicate with the server. It generates packets to be sent inside Sebek without using the TCP/IP stack and the packets are sent directly to the driver bypassing the raw socket interface. The packets are then invisible to the user and Sebek modifies the kernel to prevent the user from blocking transmition of the packets.

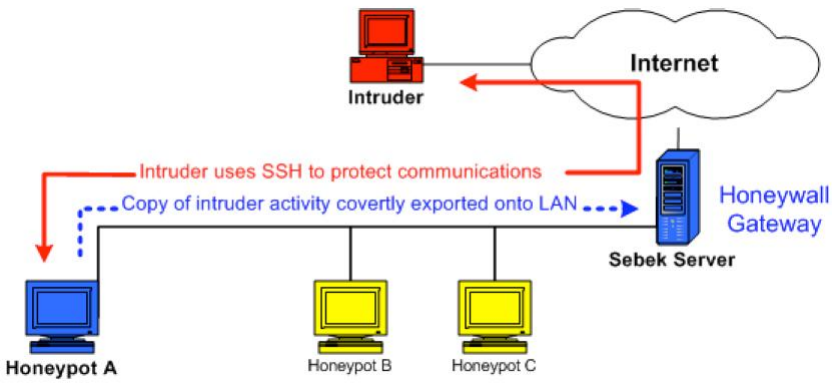


Figure 5.3: Sebek Deployment

In the case of multiple clients, there's a risk of the clients seeing each other's packets. Sebek configures its own raw socket interface on the clients to ignore all incoming Sebek packets. Only the server can receive Sebek packets. [22]

Due to its comprehensive log capabilities, it can be used as a tool for forensics data collection. It has a web interface that can perform data analysis.

5.4.3 Snort-inline

Snort_inline is a modified version of Snort. It is "an Intrusion Prevention System (IPS) that uses existing Intrusion Detection System (IDS) signatures to make decisions on packets that traverse snort_inline."[29] The decisions are usually drop, reject, modify or allow. For more on Snort, please see [30].

Chapter 6

Data Collection and Analysis

The plan was to use Sebek for data collection on the honeypots, but the web interface was not working with the Windows Sebek version. Apparently this is a problem that many have experienced according to Internet Sebek forums. Unfortunately because of the time limitations, I did not find an alternative for this. I could have started over and used another forensic tool like Sleuhkit in conjunction with Autopsy [31]. The purpose of Sebek was to detect the changes in the honeypot systems brought about by the Storm infection. The Sleuthkit tool is capable of the same. The Sebek data is supposed to be analysed on its own by the Walleye application and also used as input for snort-inline.

Because of the lack of data, this section will have to be theoretical. I can surmise what the data would have shown and perhaps another can continue the installation and fix the problems.

6.1 Expected Behaviour

From previous research, there is an expected behaviour of the Storm malware after the initial infection.

At first the binary is downloaded from a website and executed. The steps following the execution are thus:

1. A copy of the binary called spooldr.exe is created.

2. A driver spooldr.sys is installed in the Windows system32 directory.

3. The tcpip.sys driver is modified.

As an example, I executed a binary downloaded from [27] in the virtual Windows environment. I then searched for the term "spooldr" and got two results back. One was an application file and the other was a system file.

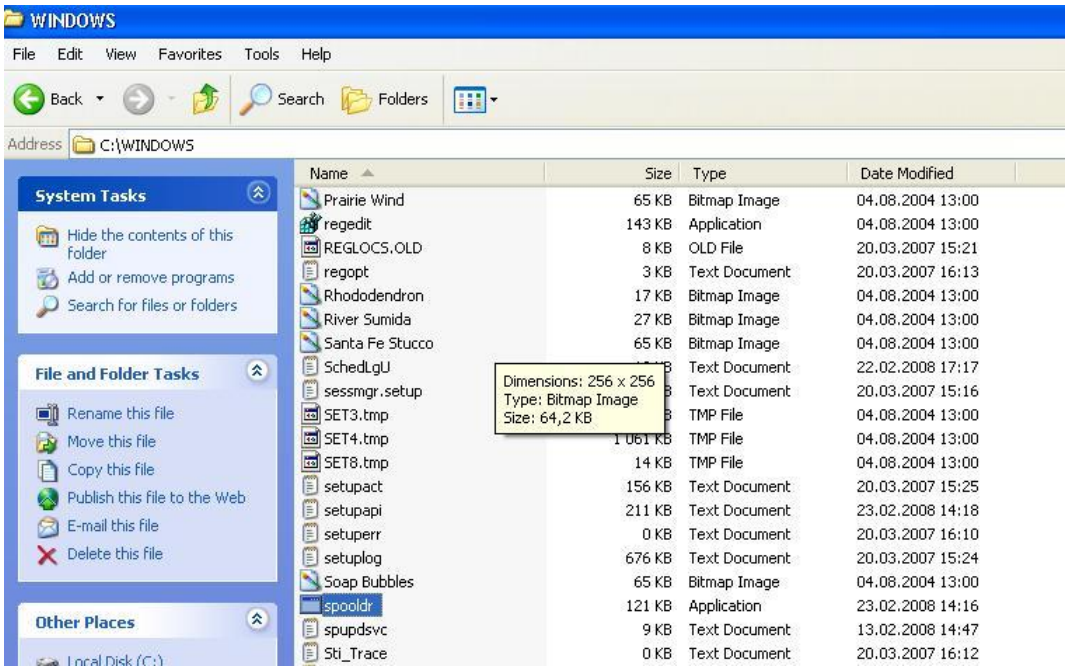


Figure 6.1: The spooldr.exe file

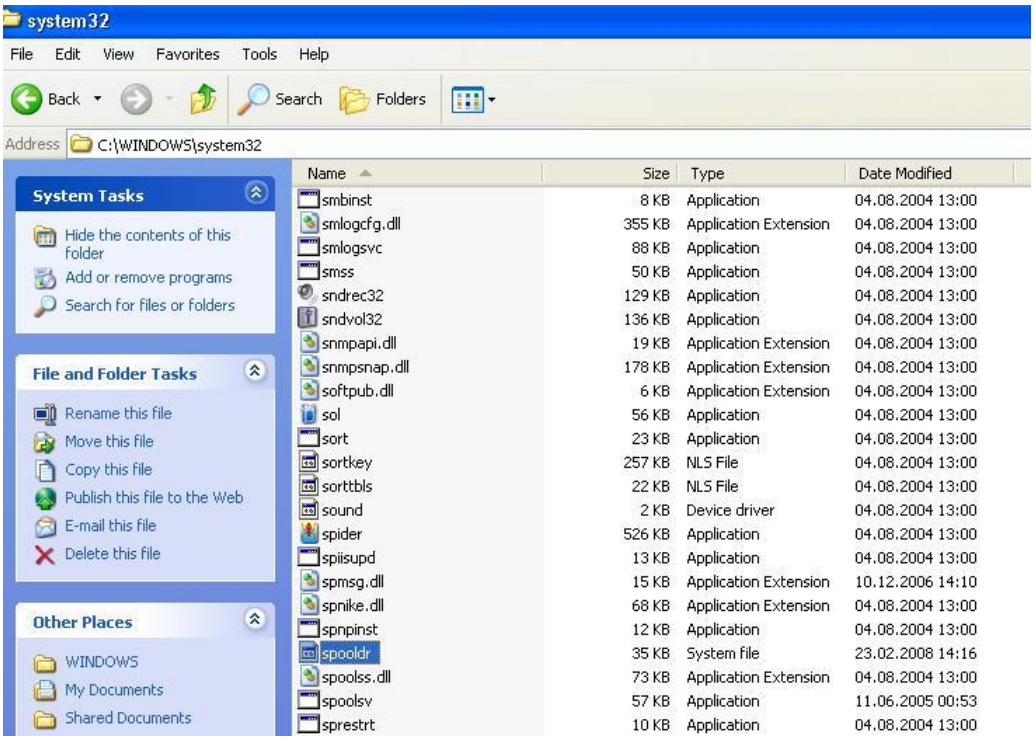


Figure 6.2: The spooldr.sys file

4. Create file spooldr.ini that contains the list of peers to be contacted. This is an initialisation file that starts the connection to the Overnet network for coordination.

At this stage, the network traffic analysis should show UDP P2P traffic, with the protocol type Overnet or Other. A closer look at the packets should reveal

the nature of the messages exchanged. If BotHunter is employed as the IDS of choice, it can identify the different dialog types (See figures in "Literature Review").

5. After a successful connection to the Overnet network, the peers in the list are contacted in bursts every 10 minutes.

6. When the peers respond, the infected machine starts the coordination part. It searches for certain hardcoded keywords and receives a an encrypted URL from the peers. The alert received looks like this:

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

Figure 6.3: Snort Peer Coordination Alert

[10]

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

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

Figure 6.4: Snort Peer Coordination Heuristics

[10]

As can be seen, BotHunter identifies the messages and Snort recognises them as a peer coordination event.

7. The URL received from the peers is decrypted and contacted. Sebek records this event and using the right timestamp, one can search for the corresponding Sebek packet. The packet would show the decryption procedure and the unencrypted URL. The IP address can also be found by looking in the IDS packets. This IP address is not worth tracking because it is most likely generated using Fast-Flux DNS, which means it will change in a few minutes.

8. Traffic monitoring between this IP address and the infected machine will show a packet download. This packet is probably encrypted. This is not a problem because the decrypted contents can be recovered through Sebek. This will be updates or an attack command.

9. The infected machine then prepares for attack. For example BotHunter recognises a potential spam attack when the machine starts collecting mail host IP addresses.

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

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

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

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

Figure 6.5: Snort Outbound Attack Propagation Alert
[10]

10. The attack itself 6.5. This is indicated by a sudden spike in outbound SMTP mail to SMTP servers in the case of a spam attack. For DDoS attacks, other protocols' traffic drastically increases. This considerably slows down the infected computer and is usually how infected machines are detected.

6.1.1 System changes

When Storm executes, several changes are made to the system files in Windows. The first of the changes is adding the new driver spooldr.sys and deleting several drivers and application files (executables). The Windows monitoring system usually sends a warning that certain files have been altered, but it does not provide any specifics. This however can be an indication that a machine has been compromised. The warning's packet contents look like this 6.6:

The image is from the Walleye application's packet analysis. It is caused by the virtual Windows machine's altered state after executing a Storm binary. Although the system was infected, it's possible that Storm detected the virtual environment because the initialisation file was not installed and the machine did not attempt to connect to the Overnet network. In a future project, it will be

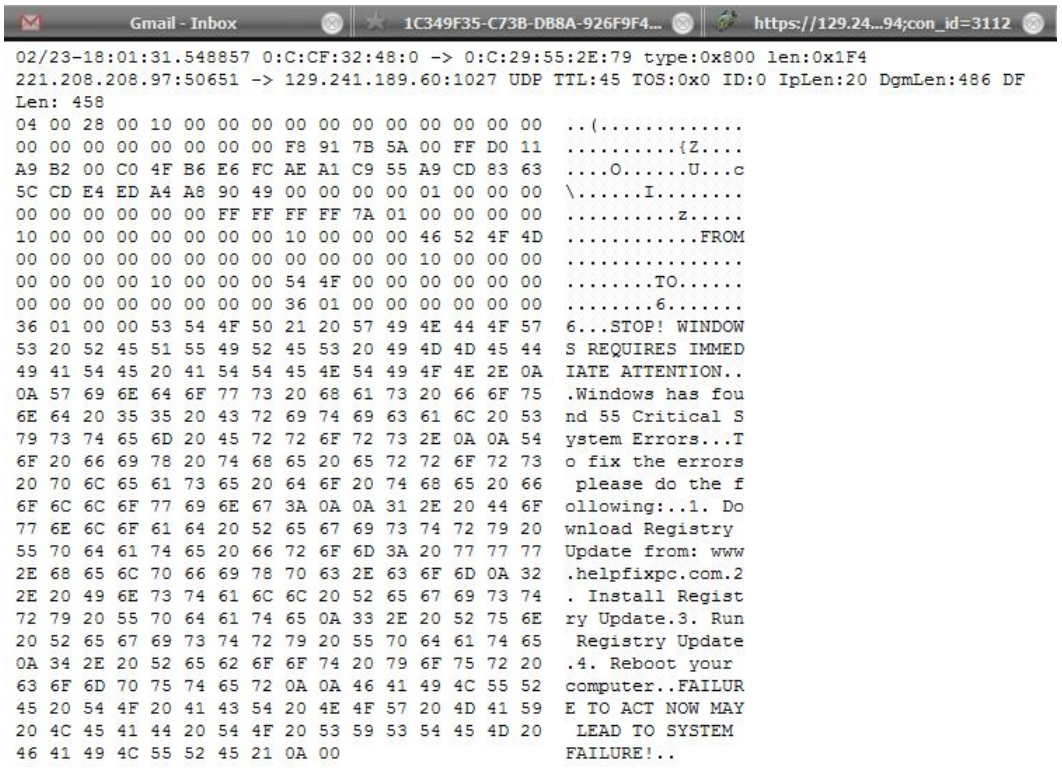


Figure 6.6: Windows Error Warning Packet

important to test a virtual Windows machine and a real Windows machine concurrently to affirm that Storm does check for a virtual environment. It would also be interesting to find out *when* it checks.

Chapter 7

Further Work

Storm is a relatively new malware. It is barely one year old and it is considered to be one of the most dangerous malwares on the Internet. It is becoming increasingly sophisticated and aggressive. So far the research has been based on finding out how it works. The next step in the research is finding out how to stop it. While prevention and detection measures are enough for most people and organisations, law enforcement agencies are interested in tracking the responsible parties and possibly shutting it down. As it is explained in this thesis, this is very hard to do for several reasons. The decentralised P2P structure makes it difficult to find a "core" that can bring down the whole network. The use of Fast-Flux makes it difficult to track their web/DNS servers. Even if the servers can be tracked, the ISP has to cooperate to be able to shut them down. Unfortunately there are countries where the laws concerning cyber criminality are so lax that prosecution is impossible. This of course is a case of "crossing that bridge when we get to it" because the previously stated problems are more pressing.

Possible future research:

1. Add BotHunter to the Honeywall. They are both opensource and it would be interesting to see the integration of the two.
2. Perform a proper forensic analysis with approved forensic tools.
3. Take a closer look at the supporting technologies and protocols: Fast-Flux and Overnet are particularly interesting.
4. Take an in-depth look at the cryptography used in Storm.

The Honeynet Project has a paper coming out soon about Storm. They also have an ongoing project on the topic. It could be a possibility to get a working collaboration with them. [7]

The ITU also has an ongoing project on Storm. It is linked to the ITU CYB project that deals with cyber crimes in general. The interesting part of this project is that they also look at the legal aspect of cybercrime.[6]

SRI International continues to work on their BotHunter tool and on Storm.[10]

Chapter 8

Conclusion

Storm worm is the first of its kind: a true P2P botnet. It is considered to be one of the most dangerous malware out there at the moment. It is notoriously difficult to detect, it has had a very successful run, it is difficult to determine its size, there are currently no means to track the botmaster(s) and it is extremely aggressive. It operates a number of websites where its victims can download binaries from. Because it uses Fast-Flux technology, the web servers are hidden behind rapidly changing DNS records and a whole horde of IP addresses. The botmaster(s) use their botnet as a shield from discovery. The use of the P2P protocol Overnet means there are no central servers or C and C server that can be tracked or shut down. Finding out that a machine is infected can be done, but it first has to arouse suspicion. This is rather different with private machines than with machines in an organisation. In an organisation, there is network monitoring and a network administrator who may notice that a certain machine has too much traffic. In a private setting, it is very unlikely that a person will think of checking their traffic flow. Because every machine is hardcoded with a peer list, more infected machines may be found that way. Even if these peers were to be disinfected, there are usually about 200 peers on a list and this would hardly even cause a dent in the Storm botnet.

Storm infects Windows machines and connects them to its botnet. The botnet is then used to send spam mail, perform synchronised DDoS attacks and other crimes.

Because the Storm binaries and messages are encrypted, they can bypass certain IDS. The Storm binaries are regularly changed, which may also prove difficult for signature based IDS. Storm worm is not easily detected, but there are tools in development that may prove to be useful, such as BotHunter. It employs a non-signature based detection method in conjunction with the IDS

Snort to positively identify a Storm worm infection. However, it works solely as an IDS. SRI made some models of the Storm behaviour and its communication patterns and used these models as input in Snort. Finding effective and widely useable means to detect Storm would be a great step and would slow down Storm's growth.

Preventative methods are few at the moment and they hinge on the human factor rather than technological. The PC users need to be aware of these sort of malwares and be more careful about what they download. Security awareness and education in organisations should be taken seriously. An alternative method would be to block all P2P protocols. This may be an unpopular move, but it will do the work. A machine may be infected with Storm, but if it can not communicate with the other peers, it is harmless. In a private setting, short of public awareness advertisements informing people about the dangers on the Internet and how to avoid them, there is little that can be done.

Lawmakers need to take cyber crime seriously and try to keep up with the technology developments. There should be clear and strict laws against writing and willfully spreading malware like Storm. ISPs should also be held accountable for whatever crimes their clients make and encouraged to cooperate to take them down. International laws would work best for the Internet because there are no geographical boundaries on the Internet and cyber criminals exploit this. Countries that allow rampant cyber crime should be pushed to reconsider their laws.

It is probable that the people behind Storm will add more complexity to it this year. They have proven to be well informed on the latest technological trends and within the year Storm has been around, it has evolved continuously. The next step is thought to be more complex cryptographic schemes, it is very likely that asymmetric encryption will be used. [32]

The situation is not hopeless though because there are new findings on Storm worm and its weaknesses made every day. Researchers, the security industry and law enforcement agencies are working on the problem. Continued study of the latest papers and publications on the topic should provide information on security measures that can be taken against Storm.

Bibliography

[1] The sans institutes top ten cyber security menaces for 2008. http://www.itu.int/osg/csd/newslog/The+SANS+Institutes+Top+Ten+Cyber+Security+Menaces+For+2008.aspx. accessed February 20, 2008.

[2] Julian B. Grizzard, Vikram Sharma, Chris Nunnery, Brent ByungHoon Kang, and David Dagon. Peer-to-peer botnets: Overview and case study. Technical report, The Johns Hopkins University Applied Physics Laboratory and University of North Carolina at Charlotte and Georgia Institute of Technology, 2007. `http://www.usenix.org/events/hotbots07/tech/full_papers/grizzard/grizzard_html/`; accessed September 30, 2007.

[3] Iain Thomson. Estonia under cyber-attack. *Computing*, 2007. `http://www.computing.co.uk/vnunet/news/2190172/estonia-under-cyberattack`; accessed February 10, 2007.

[4] Gregg Keizer. Dutch police crush big 'botnet,' arrest trio. *TechWeb News*, 2005. `http://www.informationweek.com/story/showArticle.jhtml?articleID=171204550`; accessed February 10, 2007.

[5] John Leyden. Telenor takes down 'massive' botnet. *The Register*, 2004. `http://www.theregister.co.uk/2004/09/09/telenor_botnet_dismantled/`; accessed February 10, 2007.

[6] Itu botnet mitigation toolkit. http://www.itu.int/ITU-D/cyb/cybersecurity/projects/botnet.html. accessed February 10, 2007.

[7] The honeynet project. http://www.honeynet.org. accessed February 02, 2007.

[8] Paul Barford and Mike Blodgett. Toward botnet mesocosms. Technical report, University of Wisconsin-Madison, 2007. `http://www.usenix.org/events/hotbots07/tech/full_papers/barford/barford.pdf`; accessed December 22, 2007.

[9] Massimiliano Romano, Simone Rosignoli, and Ennio Giannini. Robot wars how botnets work. *Hakin9*, 2005. `http://www.windowsecurity.com/articles/Robot-Wars-How-Botnets-Work.html?printversion`; accessed February 02, 2007.

[10] Phillip Porras, Hassen Sa¨ýdi, and Vinod Yegneswaran. A multi-perspective analysis of the storm (peacomm)worm. Technical report, Computer Science Laboratory, SRI International, 2007. http://www.cylab.cmu.edu/files/cmucylab07004.pdf; accessed October 11, 2007.

[11] Ping Wang, Sherri Sparks, and Cliff C. Zou. An advanced hybrid peer-to-peer botnet. Technical report, School of Electrical Engineering and Computer Science, University of Central Florida, Orlando, FL, 2006. http://www.usenix.org/events/hotbots07/tech/full_papers/wang/wang.pdf; accessed September 27, 2007.

[12] Reinier Schoof and Ralph Koning. Detecting peer-to-peer botnets. Technical report, System and Network Engineering, University of Amsterdam, 2007. ; accessed December 22, 2007.

[13] Gregg Keizer. Storm botnet spreading malware through geocities. *Computerworld*, 2007. http://www.pcworld.com/article/id,139736-page,1/article.html; accessed February 10, 2007.

[14] Know your enemy: Fast-flux service networks. http://www.honeynet.org/papers/ff/fast-flux.html. accessed February 11, 2007.

[15] Yi Qiao and Fabian E. Bustamante. Structured and unstructured overlays under the microscope: A measurement-based view of two p2p systems that people use. Technical report, Department of Electrical Engineering Computer Science, Northwestern University, 2005. http://www.aqualab.cs.northwestern.edu/publications/YQiao06SUO.pdf; accessed February 12, 2007.

[16] Mcgraw-hill dictionary of scientific and technical terms. http://www.answers.com/topic/computer-forensics?cat=biz-fin. accessed February 22, 2008.

[17] Colm Murphy. The rules for computer forensics. *Help Net Security*, 2007. http://www.net-security.org/article.php?id=1040&p=2; accessed February 22, 2007.

[18] Keith J. Jones, Richard Bejtlich, and Curtis W. Rose. *Real Digital Forensics: Computer Security and Incident Response*. Addison-Wesley, 2006.

[19] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. Technical report, Computer Sciences Department, University of Wisconsin, Madison, 2005. http://pages.cs.wisc.edu/~pb/botnets_final.pdf; accessed October 11, 2007.

[20] Joe Stewart. Storm worm ddos attack. Technical report, SecureWorks, 2007. http://www.secureworks.com/research/threats/storm-worm/?threat=storm-worm; accessed December 22, 2007.

[21] Linda Tucci. Fewer security breaches blamed on human error. *CIO*, 2006. http://searchcio.techtarget.com/news/article/0,289142,sid182_gci1273058,00.html; accessed February 02, 2007.

[22] Know your enemy: Sebek, a kernel based data capture tool. http://www.honeynet.org/papers/sebek.pdf. accessed February 02, 2007.

[23] Vmware. http://www.vmware.com. accessed February 22, 2008.

[24] Know your enemy: Honeywall cdrom roo. http://www.honeynet.org/papers/cdrom/roo/index.html. accessed February 02, 2007.

[25] The Honeynet Project. *Roo CDROM User's Manual*, 2007.

[26] Eirik F. G. Bergande and Jon Fjeldberg Smedsrud. Using honeypots to analyze bots and botnets. Master's thesis, Norwegian University of Science and Technology, 2007.

[27] Cyber-threat analysis research project. http://www.cyber-ta.org/malware/Malware$_{E}xamples/STORM/.accessedFebruary$22, 2007.

[28] Niels Provos and Thorsten Holz. *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Addison-Wesley, 2008.

[29] Snort$_{i}nline.http://snort-inline.sourceforge.net/.accessedFebruary$22, 2008.

[30] Martin Roesch. Snort - lightweight intrusion detection for networks. Technical report, Snort IDS, 2007. `http://www.snort.org/docs/lisapaper.txt`; accessed February 22, 2008.

[31] The sleuth kit and autopsy: Digital investigation tools for linux and other unixes. http://www.sleuthkit.org/. accessed February 21, 2007.

[32] Cybercrime 2.0: Thorsten holz. http://media.koeln.ccc.de/browse/congress/2007/24c3-2318-en-cybercrime20.html. accessed January 15, 2008.

Chapter 9

Appendix

9.1 Appendix A The Honeywall Configuration

```
# $Id: honeywall.conf 4552 2006-10-17 01:06:51Z esammons $
#
##############################################
#
# Copyright (C) <2005> <The Honeynet Project>
#
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or (at
# your option) any later version.
#
# This program is distributed in the hope that it will be useful, but
# WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU
# General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to the Free Software
# Foundation, Inc., 59 Temple Place, Suite 330, Boston, MA 02111-1307
# USA
#
##############################################
```

```
#
# This file is the Honeywall import file (aka "honeywall.conf").
# It is a list of VARIABLEVALUE tuples (including comments as
# necessary, # such as this) and whitespace lines.
#
# note: DO NOT surround values in quotation marks
#
#####################################################################
############################
# Site variables that are #
# global to all honeywalls #
# at a site. #
############################
# Specify the IP address(es) and/or networks that are allowed to connect
# to the management interface. Specify any to allow unrestricted access.
# [Valid argument: IP address(es) | IP network(s) in CIDR notation | any]
HwMANAGER29.241.208.244
# Specify the port on which SSHD will listen
# NOTE: Automatically aded to the list of TCP ports allowed in by IPTables
# [Valid argument: TCP (port 0 - 65535)]
HwSSHD_PORT22
# Specify whether or not root can login remotely over SSH
# [Valid argument: yes | no]
HwSSHD_REMOTE_ROOT_LOGINño
# NTP Time server(s)
# [Valid argument: IP address]
HwTIME_SVR############################
# Local variables that are #
# specific to each #
# honeywall at a site. #
############################
# Specify the system hostname
# [Valid argument: string ]
HwHOSTNAMElocalhost
# Specify the system DNS domain
```

```
# [Valid argument: string ]
HwDOMAIN=localdomain
#Start the Honeywall on boot
# [Valid argument: yes | no]
HwHONEYWALL_RUN=yes
# To use a headless system.
# [Valid argument: yes | no]
HwHEADLESS=no
# This Honeywall's public IP address(es)
# [Valid argument: IP address | space delimited IP addresses]
HwHPOT_PUBLIC_IP=129.241.189.55 129.241.189.60 129.241.189.40
# DNS servers honeypots are allowed to communicate with
# [Valid argument: IP address | space delimited IP addresses]
HwDNS_SVRS=129.241.200.3
# To restrict DNS access to a specific honeypot or group of honeypots, list
# them here, otherwise leave this variable blank
# [Valid argument: IP address | space delimited IP addresses | blank]
HwDNS_HOST=129.241.189.50 129.241.189.60 129.241.189.40
# The name of the externally facing network interface
# [Valid argument: eth* | br* | ppp*]
HwINET_IFACE=eth0
# The name of the internally facing network interface
# [Valid argument: eth* | br* | ppp*]
HwLAN_IFACE=eth1
# The IP internal connected to the internally facing interface
# [Valid argument: IP network in CIDR notation]
HwLAN_IP_RANGE=129.241.189.0/25
# The IP broadcast address for internal network
# [Valid argument: IP broadcast address]
HwLAN_BCAST_ADDRESS=129.241.189.255
# Enable QUEUE support to integrate with Snort-Inline filtering
# [Valid argument: yes | no]
HwQUEUE=yes
# The unit of measure for setting oubtbound connection limits
# [Valid argument: second, minute, hour, day, week, month, year]
```

HwSCALE=hour
# The number of TCP connections per unit of measure (HwScale)
# [Valid argument: integer]
HwTCPRATE=300
# The number of UDP connections per unit of measure (HwSCALE)
# [Valid argument: integer]
HwUDPRATE=20
# The number of ICMP connections per unit of measure (HwSCALE)
# [Valid argument: integer]
HwICMPRATE=50
# The number of other IP connections per unit of measure (HwSCALE)
# [Valid argument: integer]
HwOTHERRATE=50
# Enable the SEBEK collector which delivers keystroke and files
# to a remote system even if an attacker replaces daemons such as sshd
# [Valid argument: yes | no]
HwSEBEK=yes
# Enable the Walleye Web interface.
#[Valid argument: yes | no]
HwWALLEYE=yes
# Specify whether whether to drop SEBEK packets or allow them to be sent
# outside of the Honeynet.
# [Valid argument: ACCEPT | DROP]
HwSEBEK_FATE=DROP
# Specify the SEBEK destination host IP address
# [Valid argument: IP address]
HwSEBEK_DST_IP=129.241.189.1
# Specify the SEBEK destination port
# [Valid argument: port]
HwSEBEK_DST_PORT=1101
# Enable SEBEK logging in the Honeywall firewall logs
# [Valid argument: yes | no]
HwSEBEK_LOG=yes
# Specify whether the dialog menu is to be started on login to TTY1
# [Valid argument: yes | no]

HwMANAGE_DIALOGȳes

Specify whether management port is to be activated on start or not.

[Valid argument: yes | no]

HwMANAGE_STARTUPȳes

Specy the network interface for remote management. If set to br0, it will

assign MANAGE_IP to the logical bridge interface and allow its use as a

management interface. Set to none to disable the management interface.

[Valid argument: eth* | br* | ppp* | none]

HwMANAGE_IFACEēth2

IP of management Interface

[Valid argument: IP address]

HwMANAGE_IP1̄29.241.189.70

Netmask of management Interface

[Valid argument: IP netmask]

HwMANAGE_NETMASK2̄55.255.255.0

Default Gateway of management Interface

[Valid argument: IP address]

HwMANAGE_GATEWAY1̄29.241.189.1

DNS Servers of management Interface

[Valid argument: space delimited IP addresses]

HwMANAGE_DNS1̄29.241.200.3

TCP ports allowed into the management interface.

Do NOT include the SSHD port. It will automatically be included

[Valid argument: space delimited list of TCP ports]

HwALLOWED_TCP_IN2̄2 443 1101

Specify whether or not the Honeywall will restrict outbound network

connections to specific destination ports. When bridge mode is utilized,

a management interface is required to restrict outbound network connections.

[Valid argument: yes | no]

HwRESTRICTn̄o

Specity the TCP destination ports Honeypots can send network traffic to.

[Valid argument: space delimited list of UDP ports]

HwALLOWED_TCP_OUT2̄2 25 43 80 443 678

Specity the UDP destination ports Honeypots can send network traffic to.

[Valid argument: space delimited list of UDP ports]
HwALLOWED_UDP_OUT53 123 456 1101
Specify whether or not to start swatch and email alerting.
[Valid argument: yes | no]
HwALERTno
Specify email address to use for email alerting.
[Valid argument: any email address]
HwALERT_EMAILroot@localhost.localdomain
NIC Module List - Set this to the number and order you wish
to load NIC drivers, such that you get the order you want
for eth0, eth1, eth2, etc.
[Valid argument: list of strings]
#
Example: eepro100 8139too
HwNICMODLIST# Blacklist, Whitelist, and Fencelist features.
[Valid argument: string]
HwFWBLACK/etc/blacklist.txt
[Valid argument: string]
HwFWWHITE/etc/whitelist.txt
[Valid argument: string]
HwFWFENCE/etc/fencelist.txt
[Valid argument: yes | no]
HwBWLIST_ENABLEno
[Valid argument: yes | no]
HwFENCELIST_ENABLEno
The following feature allows the roo to allow attackers into the
honeypots but they can't send packets out...
[Valid argument: yes | no]
HwROACHMOTEL_ENABLEno
Disables BPF filtering based on the contents of HwHPOT_PUBLIC_IP
and the black and white list contained within HwFWBLACK and HwFWWHITE
if the HwBWLIST_ENABLE is on. Other wise, it just filters based on
the contents of HwHPOT_PUBLIC_IP
[Valid argument: yes | no]
HwBPF_DISABLEno

This capability is not yet implemented in roo. The variable
has been commented out for this reason. dittrich - 02/08/05
Options for hard drive tuning (if needed).
[Valid argument: string]
Example: -c 1 -m 16 -d
HwHWPARMOPTS# Should we swap capslock and control keys?
HwSWAP_CAPSLOCK_CONTROLño
##
Snort Rule Update Variables
##
Enable or disable automatic snort rule updates
[Valid argument: yes | no]
HwRULE_ENABLEyes
Automatically restart snort and snort_inline when automatic updates are
applied and when calls to update IDS or IPs rules?
[Valid argument: yes | no]
HwSNORT_RESTARTño
Oink Code - Required by Oinkmaster to retrieve VRT rule updates
See: /hw/docs/README.snortrules or
http://www.honeynet.org/tools/cdrom/roo/manual/
for instructions on how to obtain it (Free registration).
[Valid argument: 40 char alphanum string]
HwOINKCODE# Day automatic snort rule updates should be retrieved (for weekly updates)
For daily updates, set this to ""
[Valid argument: sun | mon | tue | wed | thu | fri | sat]
HwRULE_DAYmon
Hour of day snort rules updates should be retrieved
[Valid argument: 0 | 1 | 2 | ... | 23] (0 is Midnight, 12 is noon, 23 is 11PM)
HwRULE_HOUR12
##
Pcap and DB data retention settings
Currenrly ONLY used when Pcap/DB purge scripts are called
Pcap/DB data *is NOT* automatically purged
##

Days to retain Pcap data. This will be used *IF* /dlg/config/purgePcap.pl

is called with NO arguments.

NOTE: Override this by supplying the number of days as an argument ala:

/dlg/config/purgePcap.pl <days>

HwPCAPDAYS=45

Days to retain DB data. This will be used *IF* /dlg/config/purgeDB.pl

is called with NO arguments.

NOTE: Override this by supplying the number of days as an argument ala:

/dlg/config/purgeDB.pl <days>

HwDBDAYS=180

##

NAT mode is no longer supported.

Don't mess with anything below here unless you know what you're

doing! Don't say we didn't warn you, and don't try logging a bugzilla

request to clean up the mess!

##

Space delimited list of Honeypot ips

NOTE: MUST HAVE SAME NUMBER OF IPS AS PUBLIC_IP VARIABLE.

[Valid argument: IP address]

#HwHPOT_PRIV_IP_FOR_NAT=# Specify the IP address of the honeywall's internal (i.e. gateway

IP for NAT) IP address. This is only used in NAT mode.

[Valid argument: IP address ex: 192.168.10.1]

#HwPRIV_IP_FOR_NAT=# Specify the IP netmask for interface alises. One aliases will be created

on the external interface for each Honeypot when in NAT mode only.

[Valid argument: IP netmask]

#HwALIAS_MASK_FOR_NAT=255.255.255.0

End of honeywall.conf parameters

#

Newly defined variables as of Sat Feb 2 16:50:53 GMT 2008

#

HwHFLOW_DB=1.1

9.2 Appendix B Unrelated attacks

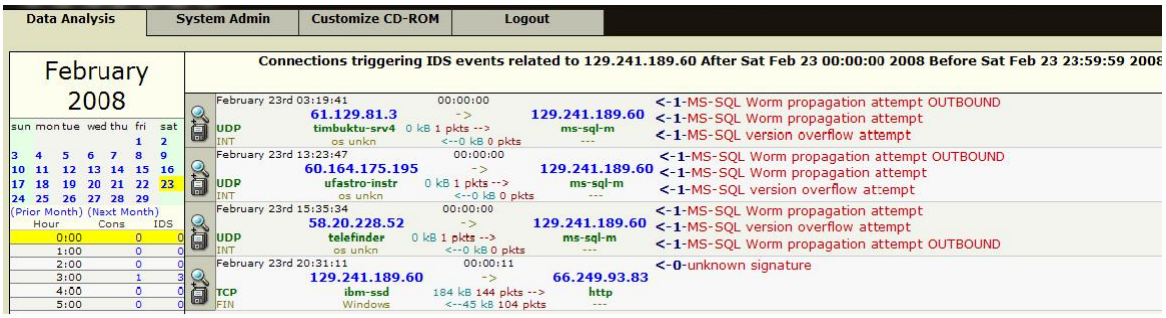


Figure 9.1: Unrelated attack attempts stopped by IDS