Final thesis

Client-side threats and a honeyclient-based defense mechanism, Honeyscout

by

Christian Clementson

LiTH-ISY-EX--09/4262--SE

2009-09-01

Final thesis

Client-side threats and a honeyclient-based defense mechanism, Honeyscout

by

Christian Clementson

LiTH-ISY-EX--09/4262--SE

Supervisor: **Stefan Pettersson**
Security consultant
at High Performance Systems

Examiner: **Viiveke Fåk**
Dept. of Electrical Engineering
at Linköpings universitet

Abstract

Client-side computers connected to the Internet today are exposed to a lot malicious activity. Browsing the web can easily result in malware infection even if the user only visits well known and trusted sites. Attackers use website vulnerabilities and ad-networks to expose their malicious code to a large user base. The continuing trend of the attackers seems to be botnet construction that collects large amounts of data which could be a serious threat to company secrets and personal integrity. Meanwhile security researches are using a technology known as honeypots/honeyclients to find and analyze new malware. This thesis takes the concept of honeyclients and combines it with a proxy and database software to construct a new kind of real time defense mechanism usable in live environments. The concept is given the name Honeyscout and it analyzes any content before it reaches the user by using visited sites as a starting point for further crawling, blacklisting any malicious content found. A proof-of-concept honeyscout has been developed using the honeyclient Monkey-Spider by Ali Ikinci as a base. Results from the evaluation shows that the concept has potential as an effective and user-friendly defense technology. There are however large needs to further optimize and speed up the crawling process.

Contents

1 Introduction **1**
1.1 Overview . . . 1
1.2 Objectives . . . 2
1.3 Limitations . . . 3
1.4 Targeted readers . . . 3
1.5 Methods . . . 3

2 Background on client-side security **4**
2.1 Client-side malware threats . . . 4
2.1.1 Virus . . . 6
2.1.2 Worms . . . 6
2.1.3 Rootkits and backdoors . . . 7
2.1.4 Bots and botnets . . . 7
2.1.5 Spyware and Adware . . . 8
2.1.6 Malicious website scripts . . . 8
2.2 Malware delivery methods . . . 9
2.2.1 Downloads and file sharing . . . 9
2.2.2 Software exploits . . . 10
2.2.3 E-mail and instant messaging . . . 11
2.2.4 Malicious and bogus websites . . . 12
2.2.5 Insecure websites . . . 13
2.3 Defenses . . . 14
2.3.1 Safe computer settings . . . 14
2.3.2 Antivirus . . . 15
2.3.3 Intrusion detection systems . . . 16
2.3.4 Website blacklisting . . . 16
2.3.5 User education . . . 17
2.4 Conclusions on client-side security . . . 17

3 Honeypots and clients **19**
3.1 Honeypots . . . 19
3.2 Honeyclients . . . 20
3.3 Existing honeyclient software . . . 21
3.3.1 Capture-HPC . . . 21
3.3.2 Monkey-Spider . . . 23

4 Honeyscout **25**
4.1 Ideal case . . . 25
4.2 Implementation . . . 27
4.2.1 Honeyclient . . . 27
4.2.2 Web crawler . . . 27
4.2.3 Programming language . . . 29
4.2.4 Database software . . . 29
4.2.5 Proxy software . . . 29
4.3 System architecture . . . 29
4.3.1 List of major code changes . . . 30
4.3.2 Blacklist database table . . . 31
4.3.3 Whitelist database table . . . 31
4.3.4 Configuration file . . . 32
4.4 User interface . . . 32
4.4.1 Heritrix . . . 33
4.4.2 Honeyscout engine feedback . . . 33
4.4.3 Blocked pages . . . 34
4.5 Limitations . . . 35

5 Evaluation **36**
5.1 Crawl scope 36
5.2 Test enviroment 37
5.3 Results 37
5.3.1 Speed 37
5.3.2 Malware detection 37
5.3.3 Usability 37

6 Future work **38**

7 Conclusions **39**

References **40**

A honeyscout.py **43**

B honeyscout.conf **46**

C monkeyscan.py **47**

D ms-scanner-clamav.py **48**

E ms-extract-arc.py **50**

List of Figures

1 Conceptual sketch 2
2 F-secure malware detection statistics (by year). 5
3 Illustration over the honeyclients operation 21
4 Illustration over Capture-HPC's design. 22
5 Illustration over Monkey-Spider's design. 23
6 Illustration over an ideal case honeyscout. 26
7 Illustration over Honeyscout's architecture. 30
8 Honeyscout's file structure. 31
9 Excerpt from blacklist database table (shortened to fit document area). 31
10 Excerpt from whitelist database table. 32
11 Heritrix's status page. 33
12 Screenshot of Honeyscout's status messages. 33
13 Screenshot of blocked page screen. 34
14 Screenshot of blocked URL inside an iframe 35

1 Introduction

A computer connected to the Internet today is more or less pre-destined to be hit by malicious traffic. Worms targeting random computers on the Internet will many times find and attack a newly connected system before it has a chance to download the latest updates. Long gone are the days when attackers only focused on servers with vulnerable services running on standard ports. One of the most exposed machines on the Internet today is a client computer running a web browser controlled by a careless human being. That's not to say that a careless person is necessarily the weakest link as there are several creative distribution methods for malware. Malicious code can for example be hidden in dynamic content served to websites via ad-networks, user-created content like forum posts or code uploaded to otherwise trusted sites via security holes in the site's software. This means that no website can be fully trusted at all times and thus makes the argument that as long as one only visits well known and professional sites you are safe, untrue. To make it even worse; weaknesses in browser software, browser plugins and operating systems makes it possible for malware to install itself on the client machine silently without the need to prompt the user and fool him/her to give it install permissions. Antivirus programs and blacklist services can constitute good protection but antivirus evasion techniques exists and blacklists are often general and might not include all sites visited by users.

What if you had a scapegoat computer, one that visits the same sites as you do, search them and follow links on those pages just like you would and then report back to you if there was any suspicious activity or malware installed when visiting any of the pages? You would simply tell it what sites you are visiting and it would follow and continue ahead, recklessly open, download and view everything it finds while monitoring its own status. All this just to warn you about malicious sites, documents and links so that you won't need to take the risk. Apply this to a bigger network with several users browsing sites at their own will all while the integrity of the internal network and the data stored there depends on a malware free environment. Maybe this kind of scapegoat computer could be a sensible addition as another layer of security in what should be a defense in depth approach to protect networks. The idea is explored in this thesis both in a theoretical way discussing the possibilities and in a practical way by creating proof-of-concept code, evaluating it and discussing problems that arise.

1.1 Overview

The idea is to investigate the honeyclient concept and the possibility to use it in a practical way as a security measure against malware hidden in content found while browsing the web.

Figure 1 illustrates how a practical implementation is thought out to work. Traffic from the clients at the internal network out to the Internet is routed through a proxy which monitors URLs visited. The URLs are forwarded to a honeyclient machine which spiders the Internet based on the URLs it receives. The honeyclient machine reports its findings to a database, creating a blacklist of URLs containing malicious code. The same database is queried by the proxy to decide if a request should be allowed or not.

There are several possible configurations, for example the proxy, database and

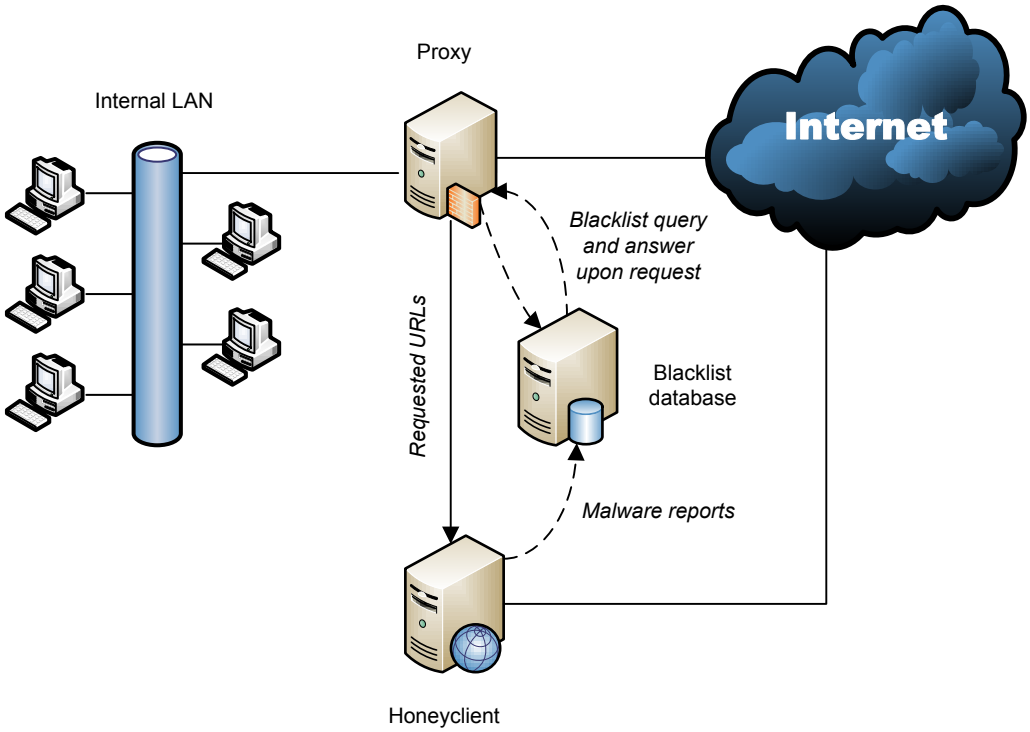

Figure 1: Conceptual sketch

honeyclient roles could all be handled by the same machine. A whitelist could be used instead of a blacklist blocking users from sites that haven't been crawled by the honeyclient yet and the proxy could allow for varied user interaction with the system. One example would be to provide the option to recrawl a blocked site for possible reevaluation and whitelisting.

1.2 Objectives

This thesis will try to evaluate and answer questions related to using honeyclient technology in a way described in the overview. Does it add something new to client security? Will it work in a live production environment? Can it be made user-friendly? How much resources are needed? It will cover several alternatives and configuration possibilities in the implementation and try to identify problems that need to be addressed. Finally it will try to give a correct indication to whether the honeyclient technology is efficient to use in this way and what needs to, or at least should be, improved to make it perform better.

The following topics of research will make up the main structure of the thesis:

- Research and explain how malicious code is distributed on the web.
- Review the different kinds of honeyclient technologies.
- Discuss how honeyclients could be used to protect users from malicious code.

- Create a proof-of-concept implementation based on the ideas from the discussion, evaluate its effectiveness and practical usefulness.

1.3 Limitations

As this is a master thesis a fixed time frame of 20 weeks is set and has to be kept. The time limitation will mostly affect the practical implementation which will be more simplistic than the considerations in the theoretical discussion. Limited time as well as limited resources will also have effect on the evaluation of the practical implementation.

Not all existing variations of honeyclients have been evaluated but the ones chosen is at the time of writing among the more popular ones and most suitable for the project.

1.4 Targeted readers

This thesis can be read by anyone interested in how malicious code is spread on the Internet and want to find out more about possible ways to discover and block such code.

Some basic understanding of the technologies behind the Internet is expected from the readers. Knowledge of Internet infrastructure and basic programming concepts are not mandatory but are recommended to fully understand the reasoning.

1.5 Methods

A lot of research has been done in the subject of IT security and malware incidents. Information is taken from books, technical papers and articles with credible sources.

2 Background on client-side security

To understand why it is interesting to use honeyclients in the way that this thesis does, some background and insight in how networks are, and used to be, attacked is needed. The attackers and their attack techniques have been changing over time, constantly trying to get past and fool the defense mechanisms set in place. It has always been a cat and mouse game, unfortunately with the attackers at the lead position making the first move. The best the defenders can do is to follow and quickly get into a position where the attackers have to make a new leap forward to succeed with their intended actions.

There are two primary attack vectors that is at first considered by an attacker while laying out his attack strategy. A computer system's functionality is a direct result of technology and humans interacting. Which one of the two an attacker decides to abuse usually depends on which one is easiest exploited at the moment. Historically attackers has preferred to exploit the technology, this makes sense because technology is predictable and lacks intelligence. Technology does not have the gut feeling that a human being does which means that it cannot separate suspicious instructions from legitimate ones. Technology does what it is told and cannot foresee or consider the consequences of actions taken. Even if a system has mechanisms to identify breaches and illegal presence on its own a human must ultimately make the decision on what countermeasures to take. This even assumes that a human has got knowledge of the incident in the first place. Thus attacking technology makes it possible to attack, infiltrate and get out before, if ever, any human can react. An over representative amount of attacks on technology leads to a big focus on more secure technology and defense mechanisms directed towards weakness' in said technology. At some point it will be more lucrative for attackers to direct their effort towards the other primary attack vector, namely the persons controlling the system. This attack vector demands different techniques, requiring more sociological knowledge than pure technological. It turns out to be fairly easy to exploit humanity, the human gut feeling is not in any way perfect and can easily be crippled by sociological factors such as stress, authority, feelings et cetera. One drawback for the attacker is that he usually only gets one good try as humans tend to get much more careful if they happen to detect an attempted attack. [1][2]

This chapter will explore the threats directed towards client computers, the techniques used by attackers to exploit Internet end users and then continue with an evaluation of existing defenses. It will not consider attacks on server-like services even if client computers may have these running and be openly exposed to the Internet. This chapter serves as a motivation to why client-side protections are necessary and why new defensive ideas must be developed.

2.1 Client-side malware threats

Reports [3] show that the point where attackers prefer to use humans as the opening attack vector instead of the increasingly hardened server technologies has come, or might even have been the case for some time. The *SANS Top-20 2007 Security Risks report* [4] indicated that client-side attacks is on the rise and at least as big a security risk as server attacks. And the trends is in favor to the attackers because not only are there a lot more client-side software than server software where vulnerabilities can be found, there are also a lot more clients than servers on the Internet to attack. Figure 2 is taken from the international anti-virus firm F-secure's *IT Security Threat Summary for*

the Second Half of 2008 [5] and shows historical detection count of malware for their signature based anti-virus product. As can be seen malware presence on the Internet have grown at an alarming rate especially between 2007 and 2008 where the amount of signatures added by F-secure tripled. Avira, another Internet security firm forecasted in a report released late 2008 [6] that year 2009 will see persisted expansion of malware threats, and the F-secure report agrees. Predictions point at botnet-construction as one of the biggest motivations for malware authors who now find opportunities to make a profit from their skills as the cybercrime world gets more organized.

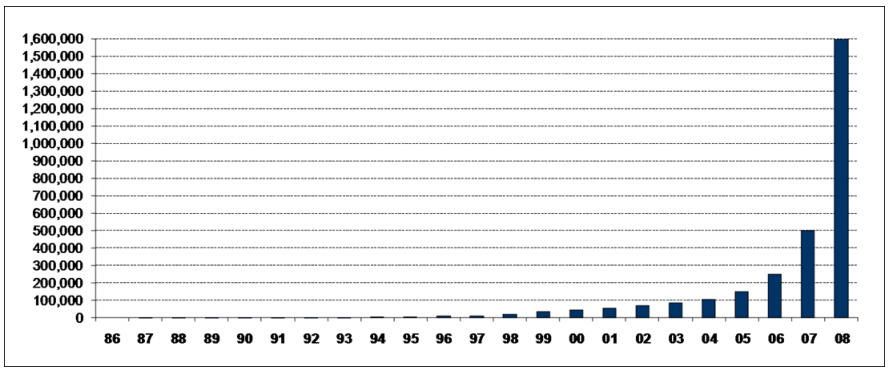

Figure 2: F-secure malware detection statistics (by year).

Client computers and client-side software refers to workstation computers with software such as a web browsers, office suites, movie viewers, et cetera, installed. Its purpose is to serve as a tool for people, one at a time, to do work on and access the rest of the network to make use of services offered by server computers. Depending on the client computer's owner it may contain everything from personal letters and photos to sensitive company information. All malicious software threats that a client computer is faced with are grouped under the same general term, malware, a abbreviation of malicious software which refers to the actual code that would perform the threatening action. An agreed on definition of malware does not seem to exist. However, in this thesis the following definition will be used:

> *Malware is a general term used to describe any code that has malicious intents such as to damage and steal data or use system resources to perform unwanted actions.*

Malware, being a general term, branches out into subgroups of more specific pieces of code all with their own characteristic malicious behavior. The rest of this section will try to describe these groups which are used by end users and security professionals alike to describe any arbitrary piece of malware code.

2.1.1 Virus

> *Infectious programs that can reproduce themselves but require interaction to propagate.* [7] (Hacking Exposed 5th edition)

One of the most widely known terms for code with unwanted behavior is computer viruses. The term *computer virus* was used as early as in the 1980s [8] and has come to represent a piece of code that attaches itself to other programs performing destructive or to the user otherwise irritating actions. This definition still lives on even if the presence of malware which purpose is solely to destroy data or to bother the user by showing dancing dogs on the screen has dropped to an almost zero. This drop is due to a shift in attitude of the malware writers which today rather write other kind of malicious code with the purpose of making money. Characteristic for viruses is that they cannot live by themselves, they need another executable program to attach itself to and it can only propagate when a infected program is run. This means that a computer virus only spreads when people are exchanging programs, which together with the fact that they have to modify executable files makes them quite easy to discover and contain.

2.1.2 Worms

> *Infectious programs that can self-propagate via a network.* [7] (Hacking Exposed 5th edition)

Computer worms are, unlike viruses, a independent program that doesn't need other executables to propagate or function. Worms instead infect the computer system itself so that they are able to live all the time while the machine is turned on. Worms can be extremely aggressive in the way they try to spread themselves, they use the network connection to exploit various vulnerabilities or send e-mails en masse in hopes of finding new victims. This aggressiveness can lead to congestion on the network and may shut down connectivity for whole organizations which ends up costing a lot of money. Because of the potential technical and economical consequences there have been a few incidents of worm outbreaks that gained mainstream attention such as the Code Red outbreak [9] in 2001 or the more modern and sophisticated Storm worm [10]. Storm got its name from the well crafted email messages it used to spread itself, using subject lines indicating the email to contain important information and video on the violent storms wreaking havoc in Europe at the time. This important (and probably emotional to many people) message made a lot of people click on the attachment and get infected.

In addition to the trouble caused by worm's propagation mechanism it usually has a purpose to fulfill on a infected machine like stealing information and e-mailing it to their creator [8]. The trend is that worms, or its spreading technique anyway, are getting more and more related to botnet creation discussed further down.

2.1.3 Rootkits and backdoors

> *Programs designed to infiltrate a system, hide their own presence, and provide administrative control and monitoring functionality to an unauthorized user or attacker.* [7] (Hacking Exposed 5th edition)

Backdoors, also known as rootkits are programs that open up the infected computer to the Internet and allow an attacker to connect back to it. The backdoor software is engineered to be hard to detect and may allow the attacker to use the computer however he wants for example to explore the file system and download files. Backdoors are mainly used in directed attacks when the attacker wants to spy on or extract specific information from a victim. Rootkit technology is also used by botnet software (further explained in section 2.1.4) to hide its presence, or more correctly, hide the bot software and its actions. The Torpig botnet [11] uses a rootkit called Mebroot, which installs itself onto the master boot record (MBR) of the victim machine. The MBR is read by the computer when it starts up, before the operating system is loaded and can therefore have a higher level of control than any antivirus software activated later. Mebroot does however not contain any of the Torpig bot functionality, it is only used to hide the Torpig files and its actions from the rest of the system and make sure the machine stays infected. Another well known rootkit technique is to load code directly into the kernel (the core of the operating system), this can be done by loading a kernel module (on UNIX systems) also called kernel hooking (Windows systems).

2.1.4 Bots and botnets

> *Very similar to rootkits and backdoors, but focused additionally on usurping the victim system's resources to perform a specific task or tasks (for example, distrubuted denial of service against an unrelated target or send spam).* [7] (Hacking Exposed 5th edition)

The word bot is an abbreviation of robot [12] and describes a kind of malware that after infection connects back to its creator (or a central command server controlled by the creator) to await further instructions. Many bots successfully connected back to their creator will make a botnet, a network of bots. Since a bot in practice have full control of the infected system a botnet could be used to complete several malicious tasks ranging from password stealing to denial-of-service attacks. Bots can have spreading techniques similar to that of worms or be hidden in executables like viruses but are in that case called trojan horses. Being infected with bot malware can have big consequences especially when illegal actions such as e-mail spamming is traced back to and blamed on the company network.

In May 2009 a group of researchers from the University of California relesed a paper called *Your botnet is my botnet: Analysis of a botnet takeover* [11] describing in detail a ten day long hijacking of the Torpig botnet. Torpig, described at the time as "one of the most advanced pieces of crimeware ever created", supplied the researchers who was in control of the central command server with 70Gb worth of data. The data included credit card numbers, valid e-mail addresses, login credentials to different services (e.g. POP3, FTP, SMTP), saved passwords as well as all data posted via input fields in web browsers. Furthermore it had functionality to serve up fake web pages displaying login screens to well known financial institutions and opened up two TCP ports on

the compromised computer allowing SOCKS- and HTTP-proxy functionality. The conclusions and potential damages resulting from a Torpig compromise where devastating. When analyzing the data the researches not only found that they had complete access to peoples e-mail and social networking accounts (among other things) they also managed to filter the data collected from web browser input fields to allow blackmailing of individuals.

2.1.5 Spyware and Adware

> *Spyware is designed to surreptitiously monitor user behaviour, usually for the purposes of logging and reporting that behaviour to online tracking companies, which in turn sell this information to advertisers or online service providers.* [7] (Hacking Exposed 5th edition)

> *Adware is broadly defined as software that inserts unwanted advertisements into your everyday computing activities.* [7] (Hacking Exposed 5th edition)

Spyware and adware are closely related to each other where adware basically is spyware with advertising functions. Spyware categorizes a type of malware that usually is installed together with other software implying it's under the users consent, but information about its presence is often well hidden and the functionality is unwanted by a big majority of the users. Spyware functionality range from sending quite innocuous statistical information about the software's usage back to the spyware company to sending sensitive information about the user and the files on his/her computer to an unknown organization. Installation of adware usually results in an increase of ads showing up while using the computer with no options to turn the functionality off. A well known spyware and adware package that infected many client computers in early 2000 was Gator who's install license (that very few people actually read) gave it permission to at any time show ads, install other software without the users knowledge and collect extensive information about the system. [13]

2.1.6 Malicious website scripts

> *Even the simplest JavaScript code snippets can do things such as pop up windows and otherwise take near-complete control of the browser's graphical interface, making it trivial to fool users into entering sensitive information or navigating to malicious sites.* [7] (Hacking Exposed 5th edition)

Most commercial and popular websites of today needs the visitor to allow client-side code to be run on their computer. This leads to more advanced web applications and better user experience but also opens up for security issues because users have to trust the visited website and are given little control over what the code is allowed to do. This functionality has not been implemented completely without thought, code from websites is often given limited privileges and more feature rich code like ActiveX and Java applications needs the user's explicit permission before it's allowed to run. However, lightweight code like JavaScript is considered harmless enough by web browsers that its run without any user permission or slightest indication that code has been run at all. In the last couple of years papers and proof-of-concept code from security researchers has

shown that JavaScript is not that harmless after all. Stealing http cookies, also known as session hijacking done through very simple scripts has been known and exploited for a long time [14]. Further progressing the possibilities are more advanced code suites such as XSS Shell [15] which is basically a JavaScript rootkit that give attackers control over the victim's web browser and allows them to for example perform key logging, extract user browsing history or launch a denial-of-service attack. So while attacks through website scripts offers very limited functionality to the attacker when desired actions such as reading and writing to the file system isn't possible the simplicity and potential quantity of users exposed is enough to make it a lucrative option to malicious minded persons.

2.2 Malware delivery methods

Before continuing to look at possible protection methods against the threats presented in section 2.1 one has to know how infection happens and what attackers do to lure users onto their malware. Because the dilemma of attacking client computers is that there needs to be an active action taken by the victim, somehow the user behind the client computer must be persuaded to execute/open a file or visit a website that the attacker has prepared. The methods used by attackers to expose their malicious code to innocent unwitting users differ depending on the situation. If the agenda is to infect as many computers as possible without any regard to which computers it is or who they belong to, an effort on broad exposure rather than design finesse is favorable. However if the attack is directed towards one or a group of significant people, convincing delivery design is more important. The creativity of attackers has at times shown to be tremendous but in other situations laziness in message design have greatly diminished the impact of the attack. As stated, a lot of focus is put on techniques where attackers try to con users into infection as it is often proven to be the easiest way.

2.2.1 Downloads and file sharing

Free downloads and file sharing networks has always been used to spread malicious code, especially viruses have been dependant on file sharing taking place. The idea is very simple, provide software that people want and they will download and run it. In practice it's more complicated, making desirable software is hard and takes time, competition from serious vendors exists in all areas and when someone finds out that the software contains malicious code word will spread quickly leading to decreased downloads. Mainly three different approaches are utilized by attackers to make infections as easy as possible:

Make simple or bogus software and try to spread it. Examples of this are screen saver software, simple games, packages with smiley characters and executable files with an interesting name but does not function. The drawback of this method is that it mostly attracts young and naive users as most people are not interested in cheap, unknown software with limited functionality. It is also hard to promote such software to a broad audience. In some situations however when the attack is directed towards a selected few this can be a good tactic, probable examples would be sending a little backdoor installing calorie counting program to the executive director that wants to lose weight or distribute password stealing online game cheat software to users who paid a lot of money for their game accounts but are not doing very well. A new take on this concept and yet another proof

of attackers creativity is the rise of *scareware*. Professional looking, free but fake anti-malware products that via online scans tells the user that his computer is infected but offers cleansing and protection if downloaded and installed [5].

Steal someone else's software, apply malicious code and give it away for free. Pirated software has been traded between strangers for as long as commercial software has existed, and a lot of trust is given to the shady suppliers of cracked software in exchange for free quality programs and games. With the recent growth of Internet communities it has become harder to use file sharing networks as a distribution channel for malware. Advanced networking features within the pirate communities that allow individuals to rate and comment files leads to quick revelation of malware infected software. At the same time trust can be given to those few that always seem to supply non-malicious downloads and continuously are given high ratings by the collective.

Collaborate with legitimate business which already has quality software that people want. This is the spyware/adware approach to malware spreading and needs some economic resources and professionalism. Because either the company owning the software needs to be convinced that the code distributed together with their software is at least somewhat legal or the real malicious functionality has to be hidden from them until it is too late. A once popular application bundled with spyware and adware, even though the company claimed that there was no such thing, is Kazaa a media desktop-application with file sharing capabilities. [16]

2.2.2 Software exploits

This category is not a distribution channel solely by itself, it is a more elegant and stealthy way of getting malware to run on the victim computer. Software exploits open up a lot of new ways for attackers to be creative in their delivery of malware and a good exploit can render users practically defenseless.

A software exploit is code that uses a vulnerability in a computer program. Furthermore, a vulnerability can refer to any technical problem or fault in the original software code that makes the program misbehave or altogether crash. One of the most popular vulnerabilities have been buffer overflows which in the right circumstances give attackers control over the computer's execution memory meaning they can continue to run any code they wish. The consequence of software exploits is that attackers aren't limited to applying their malicious code to executables using the techniques discussed in the *Downloads and file sharing section (2.2.1)*. They can by giving malformed input to a otherwise harmless application execute malicious code. This input can take many forms, for example a vulnerability in Microsoft Word can be exploited by opening a malformed Word document. To illustrate how serious software exploits can be two real world examples have been studied further.

The Microsoft ANI vulnerability found to be exploited in the wild in March 2007 gained a lot of media attention. The vulnerability existed in routines used by the Microsoft Windows operating system to process .ani files. These files contain animated cursors that for example are used to change the mouse pointer into an hour glass when the system is busy. ANI files, which mostly contain graphic content, have been used since early versions of Windows and the same old

rendering code was reused for newer versions. By loading a malformed .ani file a buffer overflow would happen and malicious code could be executed. What made this specific vulnerability critical was both the broad scope of potential victims, since the same code existed in all versions of Windows (2000, XP and Vista) everybody was vulnerable, but foremost that .ani files could be loaded very easily without the victims consent or knowledge. For example may animated cursors easily be loaded with HTML code embedded in websites or e-mails and Windows will automatically load the file when detected, it doesn't even need to have the .ani extension. What made it even worse was that example exploit code was publicly available and Microsoft was forced to quickly produce a patch. [17]

In February of 2009 a vulnerability in the Adobe Acrobat Reader software, commonly installed on client computers and used to view pdf files, was found. Yet again exploit code was made publicly available which made the situation much more severe. All versions of Acrobat Reader where vulnerable and no patch was available for several weeks. What made this vulnerability extra critical was that the malicious pdf file, which triggered a buffer overflow, didn't need to be opened by the user for exploitation to happen. Victims simply had to get the file on their computer (for example by e-mail) showing up as an icon and Adobes automatic indexing/preprocessing functions would trigger the exploit. [18]

It's no doubt that software exploits can be devastating, fortunately to some relief several variables greatly reduce attackers' ability to make use of buffer overflow vulnerabilities:

First a vulnerability has to be found, and it very rarely happens by chance. Instead attackers have to actively examine the software and try to make it crash. This takes time and requires that the attacker know what he's doing.

Then exploit code has to be written, which certainly isn't easy. It might not even be possible to use the vulnerability to write a working exploit. This process requires a lot of knowledge and expertise because if an attacker wants to use the exploit to spread malware it is not enough to get it working on his own system, it has to be compatible with all system types and configurations that he wishes to attack.

The lifetime of a vulnerability is limited, and the exploit will only work until the vulnerability is patched. If the attacker is unlucky the same vulnerability has been found by someone else and reported it to the company who releases a patch before he can produce a working exploit. This is an area where both software companies and users could do much better. Companies by getting patches out more quickly and users by actually applying the patches.

New protection technologies makes exploitation harder, non-executable memory, address randomization and system call interception have greatly increased the difficulty to write working exploits. [19]

2.2.3 E-mail and instant messaging

E-mail and instant messaging (IM) allows for direct communication with individuals many times at a semi-trusted basis where sender addresses and contact

lists tells the user from who the message originates. To the attackers delight e-mail and IM communication allows for easy attachment of arbitrary files which make them very convenient malware delivery methods. The direct communication nature of these mediums means that attackers can actively contact their victims and personalize the messages sent. To their disadvantage sending messages in the first place requires valid addresses. E-mail and IM communications can be effective both in mass malware infection attempts and directed attacks.

Mass infection attempts are often launched as spam. Spam has become the de facto expression for unwanted e-mail messages including everything from impersonal messages that promote doubtful products and stock market recommendations to newsletters from forgotten website memberships. Public knowledge about the existence of spam and the problems it may cause is relatively high since even non-technical users are annoyed by junk messages filling up their inbox. This has led to a lot of work being done to stop unwanted e-mail before it reaches the user. Yet spam doesn't seem to go away and while users have learned to be wary of suspicious e-mail messages with attachments from unknown senders most still can't resist to read and open messages that cause strong emotional reactions like the message used by the Storm worm mentioned earlier [10].

The same applies to e-mails received from trusted sources and friends, a fact that worms take advantage of when they use e-mail settings and saved contacts on a infected computer to spread itself. In the last couple of years several instant messaging worms have appeared using communication protocols such as MSN messenger, ICQ and Jabber [20]. Exploiting the trust between users can be effective way to deceive a victim that otherwise is careful enough not to fall for generic e-mail spam.

If an attacker has access to a good, un-patched software exploit e-mail and instant messaging can be highly effective distribution channels since the communication is direct, files can easily be attached and users often read the messages they get. The drawback for the attackers being that they must have some sort of list over contacts to start from.

In directed attacks personal e-mail messages can be made very convincing and while there is a general suspicion towards attachments in e-mail from unknown sources e-mail messages can be manipulated to appear to come from someone else. The threat and effectiveness from directed attacks became very clear with the release of *The snooping dragon: social-malware surveillance of the Tibetan movement* [21] a paper by Shishir Nagaraja and Ross Anderson at the University of Cambridge unveiling the attacks made against monks and sympathizers of the Tibetan freedom movement. The attacks where very well thought out and used emails that seemed to origin from trusted sources containing a convincing message in social context together with a relevant attachment (usually a PDF or PowerPoint file) to infect the victims computers. The attached file triggered an exploit when opened that installed a rootkit on the computer, from there on the attackers had full access to the victims data and could also use his/her email address and contact list to further expand their operation against members of the movement.

2.2.4 Malicious and bogus websites

Critical web browser vulnerabilities and new possibilities to write malicious website scripts have led to attackers setting up bogus websites and attempt to

lure Internet users to visit them. The quality of the attempts differ greatly, some use professional looking designs mimicking a legitimate business while other use very basic templates. The exploitation can be very quick and with techniques like drive-by-downloads practically invisible to the user after the site been visited. Drive-by-downloads refers to a technique that uses a combination of scripts and exploits to make the malware download and install itself without any further user interaction after the site had been visited.

The common problem for bogus websites is that they need users to visit it for exploitation to be possible. Sometimes accustomed Internet users can identify such sites already before they click on the link as the URL address tends to have a unusual name or prefix. E-mail spam, links posted on discussion forums or promises of interesting content that gets the site high up on search engine result lists are all ways to entice visitors. What make these sites especially dangerous is that they are fully controlled by the attackers and could contain any amount or type of code.

2.2.5 Insecure websites

In contrast to shady and suspicious websites with strange URLs high profile professional news, e-store and community sites are trusted by millions of users every day. Well known public company ownership, cryptologic certificates and social proof is more than enough for most people to feel safe and accept pretty much any content or query originating from some sites. Unfortunately attackers have found ways to spread malicious code through such sites, although more difficult, broad user exposure is guaranteed.

Malware infection trough trusted websites is done in several different ways:

Users are allowed to create content on the site and there is insufficient checks on what is created. This could be everything from posting links to malware infectious bogus sites on a discussion board to upload image files exploiting a vulnerability as profile picture.

Vulnerabilities in the web application may allow for content to be uploaded or malicious scripts to be created. Web application security is a serious concern and very few sites are completely free from security holes such as cross-site scripting that could harm the users.

Ad-networks that supplies sites with advertisements which are out of the sites control. Malware spreading through ads has been known to occur. Of special concern are Flash-based ads which could exploit vulnerabilities in the Adobe Flash player software.

The Torpig botnet infected most of its victims trough drive-by-downloads uploaded on trusted but insecure commercial websites [11]. If the site compromised has a large user base and the exploits used by the malware has good quality a drive-by-download infection approach can be highly effective. The problem with compromised legitimate websites that spread malware is larger than most people generally believe. In an interview with H-Security, Paul Ducklin, Head of Technology for the antivirus company Sophos, says that their laboratory alone finds 30,000 legitimate web sites daily that were infected with malicious JavaScript code or iframes. [22]

2.3 Defenses

Previous sections of this chapter make it seem like the Internet is a very dangerous place and being a victim of malicious code is inevitable. Although the threats are real and many computers get infected every day there is a lot one could do to reduce the risk significantly. There is no perfect solution but that doesn't mean that the current solutions are bad, it is more of a testimony to the complex nature of the human-technology relationship where technology is constantly evolving and humanity is too wide and unpredictable. Adding to this there have been an unwillingness to take on new defensive technologies and much trust is put in old concepts like antivirus and firewalls. The same kind of products that are now struggling to keep up with the new approaches taken by attackers. This unwillingness is not unjustified, cost and higher knowledge requirements are more than enough to make people decide against adopting new security products. But most of all, it is the lack of evidence that new defensive measures actually work and are more effective than what is already in use.

2.3.1 Safe computer settings

It may sound obvious that software should be developed and shipped with secure settings to make the product safe for non-technical users to use. Yet this is nothing to take for granted, because safe settings conflicts with the one most sought after product attribute, usability. If a software product wants to be user friendly it must have a lot of functions and it must allow users to use these functions easily and without obstacles. As a matter of fact most users want the software to know what they want to do and do it automatically. Especially operating systems suffers from this fact, a newly installed operating system starts with several services running and has many functions that most users will never use. Some software companies have started to understand this problem and taken various approaches to make their products more secure. One of the most successful variants has been not to reduce functionality, but to disable it and then allow for easy activation when needed. A good example is the built in firewall in Windows XP SP2 that prompt users to allow programs that wants to connect to the Internet. This way users will know which programs tries connect to the outside and at the same time, if they want the program to be able go online, easily add it to the whitelist and never be bothered again. This concept however, was not the final solution to client security. When Microsoft wanted to extend this approach in the Vista operating system it didn't receive good feedback. The idea in Vista was to limit direct access to administrative functions. A problem that had hunted Windows security for a long time was the fact that users where always logged in with administrative rights thus giving software run by the user the same rights even if it was not necessary. Users always logged in as administrators was simply a consequence of user accounts being too limited and lacked a convenient way to allow administrative actions when needed. Microsoft's solution was *User Account Control* (UAC), a security mechanism that allowed users to decide if and when a particular software could do administrative tasks. Inspired by the earlier firewall solution it simply prompted the user with a dialogue box every time a program wanted to access to restricted operating system functions giving the user a choice to whether let the program continue or cancel the action. Theoretically the idea was great, it placed the user in control of what happened on his/her machine, in practice it became close to a catastrophe. First of all software developers had come to be accustomed to their programs always having access to administrative functions which led to an unnecessary usage of such functions, in worst cases some programs needed

to get the users approval every time they started up. This combined with that most users where not technical enough to understand why they needed to be bothered all the time and what actions they were actually approving only led to irritation and enormous amounts of complaints on the Vista operating system. Consequently the next iteration of the Microsoft operating system (currently called Windows 7) will have a toned down version of UAC that surely won't be as effective.

Configuring a computer system down to sensible and secure settings will continue to be a privilege to technology savvy users for a long time until software developers figure out a balanced way to handle the security versus usability conflict. In the meantime non-technical users need help locking down their systems, because there is a lot that can be done to make client systems safer by just a few configuration tweaks which will ultimately make the whole network safer. Concrete examples would be to disable services not used, configure the web browser trusted zones settings to restrict the privileges of scripts and ActiveX/Java applets, enable automatic updates and disable autorun features.

2.3.2 Antivirus

Antivirus has come to be the last line of everyone's defense and for some, their only line of defense. There is an unjustified confidence put in antivirus products, an old concept that most attackers have learned to bypass. It is however easy to see why antivirus is so popular, it is user friendly. Antivirus software is basically a program that the user fully trusts and is given complete authority over the system, it is always active in the background scanning files and observing actions taken by other processes running. Antivirus software generally relies on signature based detection but also do behavioral analysis. It is however important that the behavior like detection mechanisms isn't too sensitive because that would give a lot of false positives and unnecessarily irritate the user when he/she is trying to do something important. This would remove the user friendliness and also the willingness to use an antivirus product at all, a fact that makes the behavior analysis mechanism less useful.

There are several problems with antivirus protection. First how broad should the definition of a virus be? As discussed before malware of today do not fit in the old virus description, detecting bot-software and rootkits may be included in the antivirus software's tasks but what about spyware and adware? Where is the line between innocent statistic gathering with user consent and information stealing? And how do the antivirus software know what the user wants and not? What about misbehavior from commercial forces like the Sony rootkit-incident where the music company Sony knownlingly distributed music CDs that installed a rootkit (without user consent) when put into a computer to prevent piracy, can users trust the antivirus software to protect them from those situations? Or will the antivirus vendors look the other way in fear of getting sued? Aside from the philosophical issues there are serious problems with the technology. A lot of today's malware can detect and bypass antivirus products, or simply disable them before the malware continues to install itself. Hiding malware in a specially crafted RAR-archive is one example on how easy it can be to evade antivirus detection [23]. Once malware have deep control of the system the antivirus can't do anything. Detection is another problem. The antivirus product needs to be constantly updated with new malware signatures but the signatures cannot be created before the malware has been found and analyzed which leaves a window of opportunity for the malware to spread and

mutate (change appearance so that the signatures won't match). This was no problem in the 90s but with today's constantly online computers and fast Internet connections it certainly is.

2.3.3 Intrusion detection systems

Intrusion detection systems (IDS) is one of the defensive technologies that have become accepted and used as a commercial network protection solution. Although it's not originally aimed at client security it works just as well for that purpose. IDSs operate by analyzing the traffic going through the network looking for malicious behavior, it is in other words trying to detect hacker attacks and malware before it reaches the targeted computer. In a big network an IDS can be a separate box connected to a tap-interface (network interface streaming a copy of all traffic on the network) thus monitoring all computers at once, this is called a NIDS (network intrusion detection system). If there is only a single computer a software IDS could be installed directly on the client and is then called a HIDS (host intrusion detection system). Having a software IDS installed directly on the computer allows for extended monitoring and protection. Because when analyzing network traffic like NIDSs does, detection have to rely on signature matching and behavior statistics but a HIDS can monitor the state of the system and activities such as data written to system folders and the addition of registry keys thus detect even stealthier kinds of malware. A continuation of the IDS concept takes the form of intrusion prevention systems (IPS) which not only detects and warns about malicious traffic but also tries to prevent such behavior when detected. It is however not always desirable to take active actions against suspicious traffic which may interrupt important events in case of false positives, therefore IDS and IPS products exists as separate entities.

2.3.4 Website blacklisting

Blacklisting is a simple security measure that has proven to be quite effective. By blocking access to malicious websites or websites with doubtful content the risk of being on the Internet has been shown to decrease substantially. Of course someone will have to build and maintain the blacklist, a very extensive and time consuming task. Building and maintaining blacklists is often done by commercial companies or security oriented organizations that offer access to their blacklist either for free, in exchange for money or as a part of their product (which for example could be a firewall solution, a proxy, et cetera). Blacklists mainly protect against the threat from bogus websites as big sites will not be put on a list by a blacklist-provider, yet as seen in section 2.2.4 such sites can also spread malware.

One of the reasons blacklists works so well is because in a lot of cases malicious scripts found on different websites all continue to download its payload from the same site unrelated to the maybe legitimate but hacked site the user was visiting [24]. One site that collects URLs for such malware snake nests is *malwaredomainlist.com* that offers its list for free to use as blacklist. The effectiveness of blacklisting is however fading, security research firm FireEye finds that attackers very well know about this weakness and are trying to eliminate it by instead store the whole payload on the hacked domain [25]. It is also possible to make blacklists obsolete by allowing botnets to serve the payload using fast-flux techniques (address to the payload is constantly changing between individual computers in a botnet).

2.3.5 User education

Ever since the beginning of computer technology, computer crashes, random system faults, lost data and undesired system behavior have been blamed on user inexperience by technology savvy people. There may be truth to some of those accusations but often technology is just as much at fault. The benefits of user education and the question if it really has any effects at all is well debated among security experts [26]. The majority side seems to be the one claiming that user education has little to no effects in raising security. A good example is the password problem, it has been said for decades that choosing a good password is important, several easy suggestions on how to make a password more complex and harder to guess are often given at the time of password choice, still weak and easy guessable passwords are a big problem. A reason a lot of malware attacks today are so simplistic and easy to spot for a well informed person is because they work well enough anyway. Yes, there is probably some attacks that could be avoided by teaching people to not being gullible on the Internet but it won't be a solution to the overall problem, this is because the average computer user will never fully understand how a computer works and thus the attacker will always have an advantage. Take for example the Tibetan monks, which very well knew they were targeted and spied on by resourceful people. Still they could not prevent being victims of attacks using social malware (a term coined by the people behind the research referring to malware that is spread using social elements), and no education could probably ever prevent it [21].

2.4 Conclusions on client-side security

This chapter has discussed existing client-side malware threats, the attacker's methods to implement these threats in reality and what security measures are generally taken to protect client computers. The complex relationship between humanity and technology have also been touched upon and the conclusion drawn is that both need attention to raise the overall security, because even if it was possible to solve technology issues like buffer overflows it will never be enough. Raising users awareness of how malware spreads on the Internet may help a bit on the way but isn't the ultimate solution. In the end technology has to bear most of the burden and take its role as assistance provider for humans. In turn technology needs humanitarian help to advance, both in new defensive techniques and in its communication with humans so that both parts better can understand each other.

The trends of malware are quite clear, successful techniques form old concepts are used with new ideas to build botnets. Botnets and its authors are more effective and strive financial gain instead of fame. Users are no longer bothered by noisy or badly written malware that makes the system malfunction. Instead all of their data, passwords, financial information and personal communication is stolen silently if front of their eyes, and may be sold to the highest bidder. Lost data or lowered productivity is not the biggest threat anymore, financial loss or public humiliation may be much worse for individuals and organizations alike. Part of the problem is that users do not grasp what is going on and the consequences of their data getting into the wrong hands. Educating users on how malware act and what data is stolen may not help computer security but hopefully make users more careful and aware when interacting with computers and the Internet.

Looking at it from another view a lot of the security issues can be solved by

making computers less general purpose. If a computer used to save and edit important documents didn't have a web browser or e-mail client installed those documents would be much more secure. If online banking applications where used from a separate closed gadget, stealing login information by installing malware on the device would be close to impossible. Maybe this kind of application isolation will be widespread in the future but today's reality is far from it. Until some technological breakthrough that revolutionizes computer security happens, new defense methods relating to today's technology is needed to keep up with the attackers. The next half of this thesis will present ideas and conceptual descriptions of a defensive measure that could be used to raise the security in networks where users are allowed to access and browse the web, thus exposing themselves to malware threats.

3 Honeypots and clients

The idea behind honeypots is deception, a tactic that has been used to catch thieves and other malicious people troughout history. A honeypot exists to lure attackers to it like bees to a pot of honey, just as its name suggests. Its function is also to keep the attacker occupied and away from the important systems. One could say that a honeypot should be as sticky as possible just like honey. As an attacker in the real world, a thief for example, would choose to break into the big exclusive mansion where no one seems to be home rather than the plain and simple family villa which has the kitchen light turned on. So would an attacker on the Internet choose the easiest target with the biggest payoff to the lowest risk. If the mansion is large enough and valuable objects seems to be hidden behind every closet the thief might be occupied just enough for the silent alarm to notify the police and allow for an onsite arrest.

It is not hard to understand why they are called honeypots but to define what it is tends to be harder. In his paper *Definitions and Value of Honeypots* [27] Lance Spitzner, a honeypot researcher and author of several whitepapers in the subject, makes the following definition of honeypots:

A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource

It may seem like a very general definition but the truth is that honeypots exists in many various forms and they can be very flexible. What they all have in common however is that their value increases the more they get abused by users with malicious intent. It is important that honeypots does not have any function at all important to the rest of the system. Neither should it have any authority to influence the system's real functionality. [27]

Honeyclients are a logical evolution of honeypots following the trend of the attackers to exploit clients and human factors instead of the servers. It's good to have some basic knowledge of honeypots and their purpose before continuing with honeyclients. Therefore a brief introduction to honeypots is included in this chapter. The rest of this chapter will explain the inner workings of honeyclients and study existing honeyclient software.

3.1 Honeypots

There are two general categories of honeypots, low-interaction and high-interaction. Low-interaction honeypots are used in production environments as a security measure to protect real networks. High-interaction honeypots are generally used by researchers to study attackers' behavior and find new software vulnerabilities and worms. They are both separate systems on the network posing as important servers with open ports of known services that would seem interesting to an attacker. In an effort to really lure attackers to choose the honeypot server instead of the real production servers the services running could be older versions of the software with known vulnerabilities. [27]

Low-interaction honeypots usually consist of a locked down unmodified operating system install with honeypot software running on top. The software can be a series of scripts listening on ports acting as real services but does not function beyond the initial connect and login routines. This makes low-interaction hon-

eypots easy to setup and use, but also easier for attackers to discover because extended functionality is missing. [28]

High-interaction honeypots are fully working systems running real services fully connectable and exploitable by attackers. Instead of honeypot software-scripts running in the background alerting the admin about break-in attempts the operating system itself are modified at the core. Much like a rootkit the modified operating system silently logs every action taken, almost undetectable by the attacker. This way much more information of the attack can be collected and the attacker can be kept busy for a longer time before he realizes he's been fooled. The drawback being that high-interaction honeypots are much harder to setup and maintain, just restoring the system every time a successful compromise has taken place can be time consuming.

3.2 Honeyclients

As a result of better server security and mature server software which has been around for so long many of the obvious vulnerabilities have been ironed out attackers have directed their attention towards client-computers. Honeyclients is one of the steps taken by security researchers to keep up with the attackers. It is the honeypot concept taken to the client-side landscape and the ideas are the same overall. The big difference is that while honeypots sits passively waiting to be attacked honeyclients have to actively search for and expose itself to potentially malicious content. This is illustrated in figure 3. Furthermore this leads to the problem that unlike honeypots which can label any traffic to itself as malicious, honeyclients has to be able to differ legitimate data from harmful. [24]

Since it is up to the honeyclient itself to find malicious content an important part is the seed mechanism which is the part of the honeyclient that finds websites to visit and files to download. There are many ways for a seeder to function, usually it starts at one or more sites which it is given manually and then it continues to spider (or crawl) to other sites from there. This means that it follows the links it finds on those sites, the next sites and so on. Some honeyclients can use search engines such as Google or Live-search to find interesting links to start from, this way different keywords can be taken as input and sites with a common theme will be crawled.

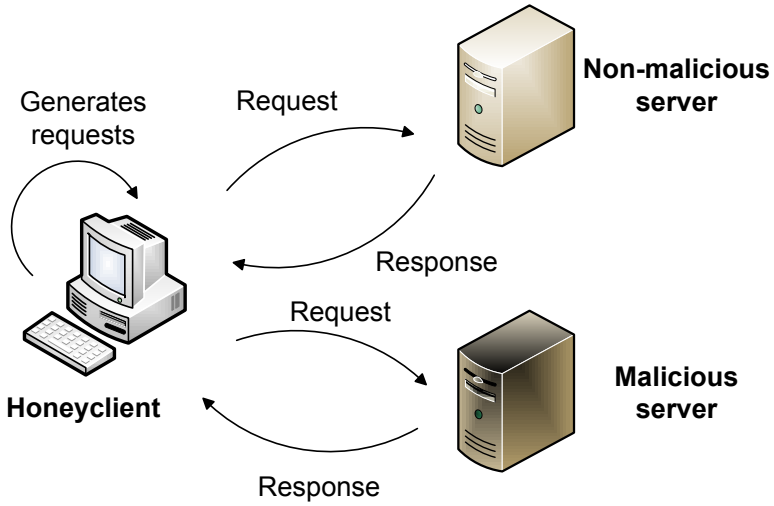

Figure 3: Illustration over the honeyclients operation

Just like in the honeypot world honeyclients are divided in low-interaction and high-interaction variants. Low-interaction honeyclients usually just download content and then try to analyze it through signature matching and other static analysis methods. This makes the machine running the honeyclient software safe from exploitations since no malicious code is actually executed, but has the downside that a lot of unknown malware cannot be detected. High-interaction honeyclients on the other hand executes and opens all files downloaded with real client-side software and additional plug-ins while being monitored by a rootkit looking for malicious actions on the system.

3.3 Existing honeyclient software

There are several existing honeyclient software packages, both commercial and free. For the purpose of the project done together with this thesis two potential candidate honeyclients where chosen. Both candidates are open source for the obvious reason that they are freely available and allows for necessary changes to the code to be made [29][30]. One is a high-interaction honeyclient while the other is low-interaction.

An updated list of existing commercial and open source honeyclient software can be found in the Wikipedia article about honeyclients [31].

3.3.1 Capture-HPC

Capture-HPC is an high-interaction honeyclient developed by Ramon Steenson and Christian Seifert at the Victoria University of Wellington together with the New Zealand Honeynet Project [32]. It is based on VMware virtual machine (VM) and server software (VMware Server) which is freely available from the VMware homepage [33]. To make it scalable it is build around a client-server infrastructure which means that several honeyclient-VMs could be run on different computers all being controlled by one computer running the server software. The control software is written in Java and utilizes the VMware API to control what actions the honeyclients take. If any VM is compromised by malicious

code it can simply be reverted back to a clean state with the VMware software's snapshot function.

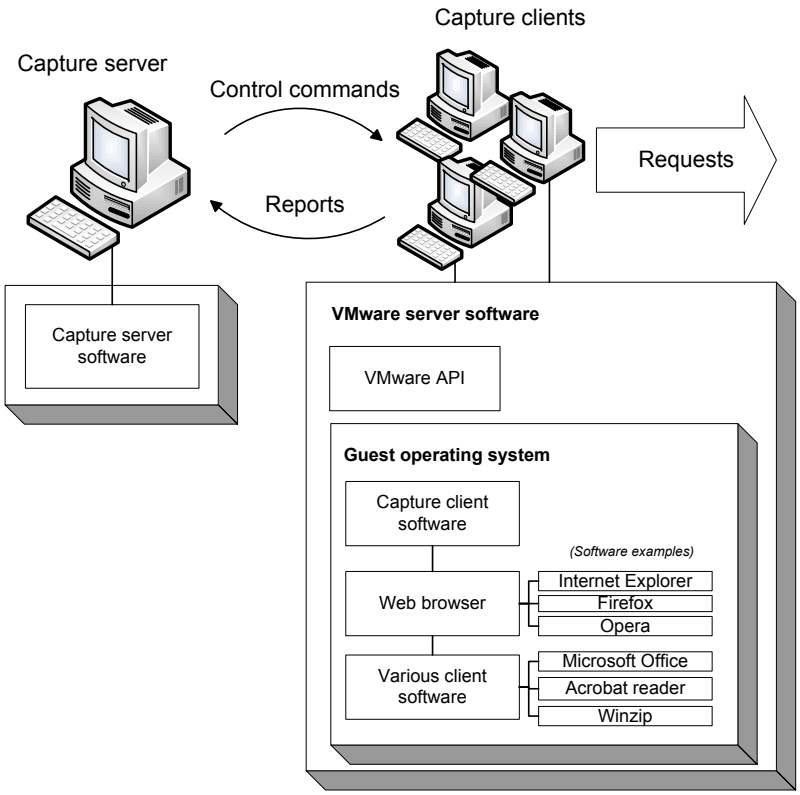

Figure 4: Illustration over Capture-HPC's design.

The monitoring part of the software which also exists as a standalone project at the New Zealand Honeynet website under the name Capture-BAT [34] is a set of kernel drivers which monitors the file system, registry, and processes that are running. In practice this means that every new process started, registry key added and file written on the running system is reported back to the Capture server. While these are actions taken by all malware at some point to infect a computer, many legitimate processes act in similar ways. To be able to separate legitimate actions from malicious ones a whitelist mechanism exists where known legitimate actions taken by the operating system can be added. A readymade whitelist with actions taken by a Windows XP SP2 system while idle is included in the default install package of Capture-HPC. [24]

The client-server model and use of VMware's API makes Capture-HPC a very flexible honeyclient. Any software, such as Microsoft Word, Adobe Reader or WinZip can be installed on the virtual machine system and used to open files found for example in e-mails or while browsing the web. The Capture-HPC server software controls which actions are taken by the honeyclient and provides it with website links to visit. If any malicious activity is reported back to the Capture server it logs what action triggered it (e.g. opening a PowerPoint-file), what kind of malicious activity took place (e.g. a new file written to the system folder), resets the infected VM (using the VMware snapshot functionality) and continues with new instructions. While VMware could handle almost any operating system the Capture-HPC system is limited to Windows XP SP2 and Windows Vista clients. This is because of the system monitoring part that

has to be customized for a specific operating system version. In the later half of 2009 version 3.0 of Capture-HPC will hopefully be released. The new release will have extended functionality such as database integration and network monitoring [35]. This functionality, especially the database integration will make it even more favorable to use in an automatized way.

3.3.2 Monkey-Spider

In 2007, Ali Ikinci, a student at the University of Mannheim, Germany finished his master thesis *Monkey-Spider: Detecting Malicious Web Sites* [12] the result was a software and script package called Monkey-Spider [36]. Monkey-Spider fits into the definition of a low-interaction honeyclient, it uses a web crawler called Heritrix [37], an antivirus/malware software package called ClamAV [38] for signature based detection of malicious code and a series of Python scripts [39] to make them work together. Figure 5 shows Monkey-Spider's different parts and how they connect. Instead of a reporting feature which just outputs text files a database entry is added for every malware found together with time and the full URL to where the malware was found.

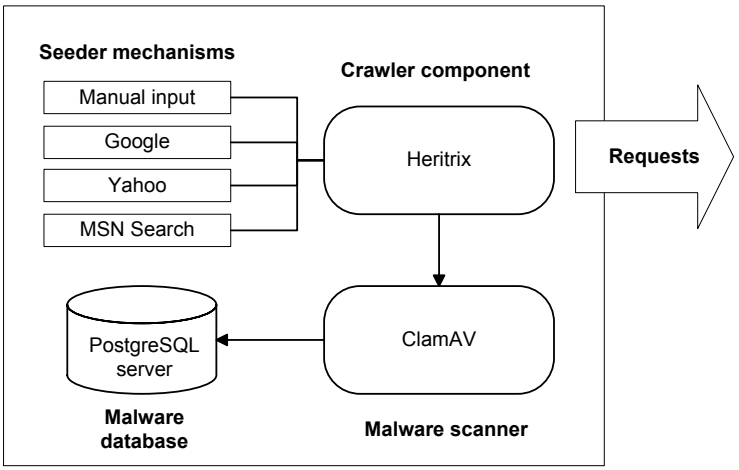

Figure 5: Illustration over Monkey-Spider's design.

Heritrix is a webcrawler developed for *The Internet Archive* [40] which downloads, saves and indexes websites over time and makes them accessible to the public through the *Way Back Machine*. Heritrix is a very extensive and flexible crawler with a lot of options ranging from different crawling techniques to content filtering. It comes with a user-friendly web interface where different crawling profiles can be created and interesting job statistics be viewed. Heritrix's main purpose is to dump whole websites with content such as scripts in an uninterpreted state and save it all in a manageable archive file-format called ARC.

Being a low-interaction honeyclient Monkey-Spider does not have the same ability to study malware behavior as high-interaction honeyclients like Capture-HPC has. It is completely dependent on the ClamAV engine to differ legit content from malicious and thus only detects known malware for which ClamAV has signatures. Theoretically the detection functionality could be extended, phoneyc [41] for example, another low-interaction honeyclient uses Javascript and visual basic script engines to find and analyze malicious Javascript and visual basic code respectively. Also other honeypot/client software is known to

use libemu, an x86-cpu emulator to find, execute and analyze shellcode. These extensions are not planned to be implemented in Monkey-Spider. However there is a possibility to use CWSandbox [42] for malware analysis but it is not available in the released version and a paid-for license of CWSandbox seems to be necessary.

4 Honeyscout

Now, the background chapter has laid down a foundation of knowledge of how client computers on the Internet are threatened by attackers and their methods. Also discussed are the defense mechanisms in use today and why they are inadequate. Further, chapter 3 introduced the honeyclient concept and how such software generally functions. It is now time to tie it all together while returning to the idea presented in the introduction and give answers to the questions stated in the objectives section.

The first thing to understand is that the outcome of this project is supposed to represent a new step in defensive technologies, in this context it will be the third and final step. The first step was the post-infection defensive capabilities of antivirus that could scan your computer to find files infected by viruses and hopefully disinfect them. The second step is the in-the-moment technologies such as intrusion detection systems and the newer generation of antivirus products that analyze data and finds malware in real time. It is now time for the third step, a defense that acts pre-infection and prevents the malware to even being able to come close to any clients, no matter if the malware signatures are known or not. Furthermore the concept is meant to operate in networks with at least a handful of client computers connected. It is doubtful that this kind of defense is viable and economicaly defendable for single computer users. However, it is not improbable that a different implementation aimed at stand-alone clients can be successfully developed.

From this point on defensive solutions based on the concept of using a honeyclient, proxy and blacklist database as described in section 1.1 will be called *honeyscouts*. The name honeyscout has two meanings to it. The first part of the name, *honey*, declares that the concept is related to the rest of the honeysoftware family (i.e. honeypot, honeyclient et cetera). The second part, *scout*, refers to how the software function as it scouts areas of the Internet ahead of the users. The definition of a honeyscout will be:

> *A honeyscout is a separate, non-critical system on a network that monitors and follows users' Internet usage scanning for and blocking malicious content to prevent client infection.*

4.1 Ideal case

Before going into the developed practical implementation, a theoretical ideal case implementation will be discussed. This is to explore the full potential of the concept as well as bring up obvious problems and how they could be handled. An ideal case of a honeyscout (illustrated in figure 6) would be based upon a high-interaction honeyclient that as closely as possible resemble the systems it is supposed to protect. This means that a honeyscout based upon, for example, Capture-HPC would run the same operating system as the clients. It would also have the same programs, updates and plugins installed configured to open files the way users would open the (i.e. the same file associations). This is possible in a network with a homogeneous client structure, such as many business and office networks that only allows a specific set of software to be installed. Smaller networks with individual client configurations might be harder to pin down but it is still perfectly doable to have the honeyclient run a standard set of software that is likely to be exploited. A high-interaction type of honeyclient would also

allow for very flexible detection of malicious behavior. In the case of Capture-HPC which monitor file system activity, registry edits and running processes it is possible to make case-by-case definitions of what malicious behavior is. An administrator could for example deny access to any website that serves content which invokes file writes to any part of the hard drive except the web browser cache.

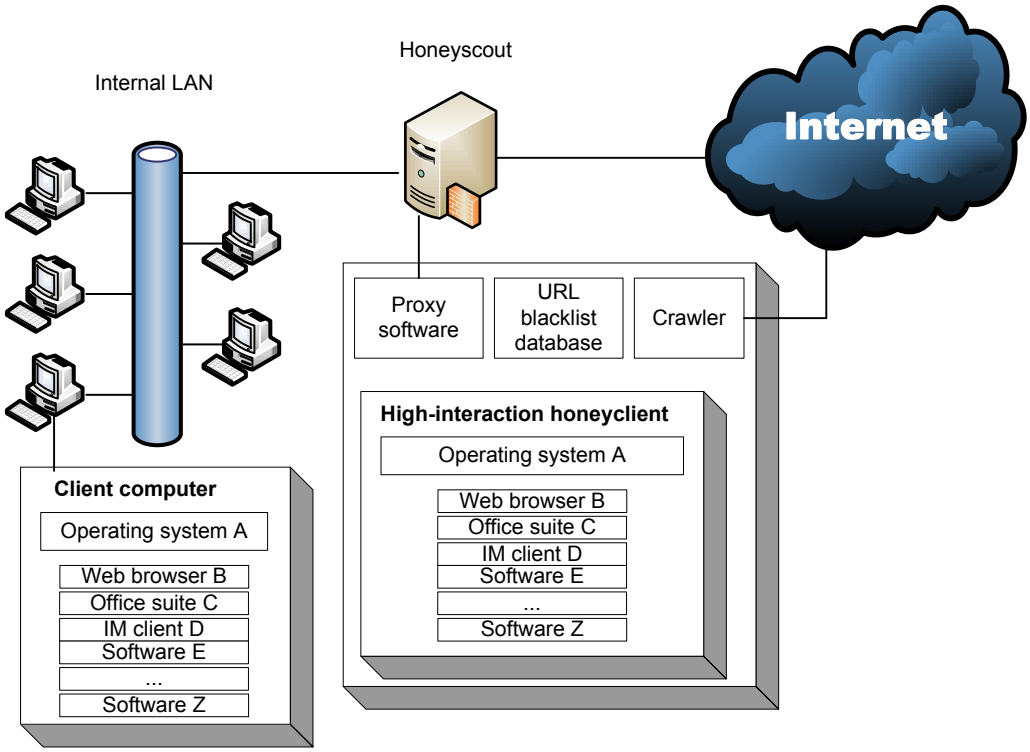

Figure 6: Illustration over an ideal case honeyscout.

With this ideal case in mind some statements of what honeyscouts potentially can do to improve client security follows:

It can detect any known or unknown malicious code that tries to influence the system. High-interaction honeyscouts are not dependant on detection signatures as it is behavioral analysis dependant. It does however not hinder the user from executing important tasks as a HIPS would do because the analysis is taken place on the honeyclient system and only protects against malicious code coming from the open Internet. This means that installation executables and similar media can still be distributed to the clients by other means.

Malware distributed through an otherwise trusted site can still be detected depending on how often the honeyscout is configured to rescan the content.

By being able to dynamically change the detection rules the protection can be adjusted to suit the current situation. A honeyscout could be configured to be very paranoid for a period of elevated threat level and then adjusted back to more permissive settings when the threat is over, it could

also be configured to block specific threats, for example to block content that start up certain processes.

Other untrusted Internet traffic may also be scanned, for example e-mail attachments which is often used in attacks.

There are also some potential problematic issues that come to mind:

It does not protect against every threat, for example malicious website scripts that do not invoke any activity on the system. Web site vulnerabilities such as cross-site scripting that could be used to steal user input would go by a honeyscout undetected.

There might be some privacy issues, much like the concerns regarding caching proxies. There is however no need for a honeyscout to log which client did a specific request.

Bandwidth usage will rise, due to the nature of web crawlers the honeyscout probably is going to crawl more content than the users request. One solution to this problem could be that the concept is combined with a caching proxy. This would also decrease web content loading times for users.

4.2 Implementation

Implementing an ideal case honeyscout would take a lot of time and resources. To make it possible to develop a working proof-of-concept honeyscout over the thesis time period a lot of compromises had to be done. The goal has been to prove that the concept would work and what kind of problems show up in practice so that future honeyscout developments have an idea of what to expect. The project will be referred to as Honeyscout and source code and database structure can be found in the thesis appendices. Honeyscout will be limited to intercept and analyze HTTP traffic as the proxy will work as a webproxy only.

4.2.1 Honeyclient

Honeyclient of choice ended up to be Monkey-Spider, developed by Ali Ikinci and introduced in section 3.3.2. There are several reasons for this decision; first of all it is a light-weight software package that is easy to understand. It is also modular which makes it easier to modify the different parts. Furthermore it already uses a database to store malware reports and it is written in the scripting language Python which is the preferred language of the author. Version used for the project is 0.2, released 24th of March 2009.

4.2.2 Web crawler

Being a part of Monkey-Spider honeyclient, the web crawler function is such an important part of the process that it calls for a closer study. Mentioned in section 3.3.2, Heritrix, which has a solid developer and user base tied to *The Internet Archive* is what the Monkey-Spider project's author decided to use. One of the foremost reasons (stated in Ali Ikinci's thesis [12]) the choice came to be Heritrix was the way it captures the unchanged state of a web site which allows for beneficial detection of malware. Further the link extraction and

queuing mechanism functions of Heritrix is far more extensive and flexible than the alternatives. The web interface and its status page might not constitute any vital functions when it comes to Honeyscout operation but is definitely a nice bonus. Section 4.4.1 further reviews Heritrix's user interface.

Some of the questions that arise when one wants to crawl a web site is, which content to crawl? How to extract links from the crawled content? How deep (far) to crawl? Heritrix offers delicate options to control the behaviour related to those questions which also will play an important role when experimenting with the honeyscout concept. The crawl decisions are controlled in Heritrix with the help of modules. Several modules may be loaded in different order to achieve desired behavior and users can create and load their own modules if the original ones are inadequate. The following crawl scope modules (e.g. what links to follow and in what order) come with the Heritrix installation and in combination offer flexible crawl rules that are sufficient for most tasks.

BroadScope - This scope allows for limiting the depth of a crawl (how many links away Heritrix should crawl) but does not impose any limits on the hosts, domains, or URL paths crawled. [43]

DomainScope - This scope limits discovered URLs to the set of domains defined by the provided seeds. That is any URL discovered belonging to a domain from which one of the seed came is within scope. [43]

HostScope - This scope limits discovered URLs to the set of hosts defined by the provided seeds. [43]

PathScope - This scope goes yet further and limits the discovered URLs to a section of paths on hosts defined by the seeds. Of course any host that has a seed pointing at its root (i.e. www.sample.com/index.html) will be included in full where as a host whose only seed is www.sample2.com/path/index.html will be limited to URIs under /path/. [43]

SurtPrefixScope - A highly flexible and fairly efficient scope which can crawl within defined domains, individual hosts, or path-defined areas of hosts, or any mixture of those, depending on the configuration. [43]

DecidingScope - A highly configurable scope. By adding different filters in different combinations this scope can be configured to provide a wide variety of behavior [43]. This scope is supposed to replace all of the above mentioned scopes by instead implementing their functionality as filters. Thus the different behaviors of the scopes described are still relevant and will be evaluated during Honeyscout configuration.

In addition to the different scopes (filters) several values can be set to further restrict and control the crawl. Examples of such values are maximum number of links to follow, maximum file size for downloadable content or time to wait before a request times out.

Since Heritrix is developed in Java almost all options and actions accessible from the web interface can also be reached via a jmx-client to directly change variables or start a new crawl job. JMX stands for Java Management Extensions and is an interface to the Java virtual machine that makes it possible to modify the state of the running program. Honeyscout uses the jmx-client that comes with Heritrix to start new crawls.

4.2.3 Programming language

Python is a dynamic object-oriented programming language that is used for many different programming tasks. It is known for its simple syntax and extensive standard library that can handle for example compression of files or URL parsing. The wide range of functionality has led to Python users coining the phrase *a programming language with batteries included*. Python is often used to create smaller scripts that does a specific task but can also be used in bigger programs. Examples of competitors to the Python language would be Perl, Ruby and Java. [39]

4.2.4 Database software

The Monkey-Spider honeyclient uses PostgreSQL as its database software. PostgreSQL is BSD licensed software that can be used and modified freely at no cost. The software and its performance is considered to be of good quality and PostgreSQL servers can often be found running in commercial production environments. This project will only use a very basic set of features available but since speed and reliability is still important PostgreSQL is a good choice. [44]

4.2.5 Proxy software

The only component that is missing from Monkey-Spider that would make it possible to modify it into a honeyscout is a gateway that can intercept the outgoing requests; any simple proxy server software could manage this task. A proxy server is a server that sits between the internal network and the Internet. When a client connects to the proxy server, requesting some service, for example a web page the proxy server evaluates the request according to its filtering rules and if accepted mediate the traffic exchange. There is no need for any advanced functions, more important is that the software is easy to modify and released under a license that allows modification. Searching the Internet one finds a lot of open source HTTP proxies to choose from, most suitable for the project came to be *Tiny HTTP Proxy*. Written in Python, very basic and released under the MIT-license Tiny HTTP Proxy is ideal to use in this case.

Tiny HTTP Proxy is written by Hisao Suzuki and was first released in 2003 on the python mailing list but now has a homepage where it can be downloaded along with the license [45]. The size is small, only a little more than 120 lines of code, and because it is written in Python it is very easy to consolidate with Monkey-Spider.

4.3 System architecture

Figure 7 illustrates how Honeyscout's internals work. The Monkey-Spider scripts that tie Heritrix, ClamAV and a database together into a honeyclient have been slightly modified. Additionally the Tiny HTTP Proxy script has been extended with new functionality and constitutes the base of the Honeyscout startup script. With the Monkey-Spider honeyclient the scripts have to be manually run after each other to complete the process, Honeyscout's startup script makes everything automatic. The complete Honeyscout file structure is shown in figure 8.

To avoid multiple seeding of the same URL a whitelist database was created. Every time a new URL is sent to Heritrix it is also added to the whitelist. A

full list of modifications and database additions follow below. In figure 7 the whole honeyscout runs on the same machine, just as with the real test system used during the project. It would however be possible to spread the different components over several machines.

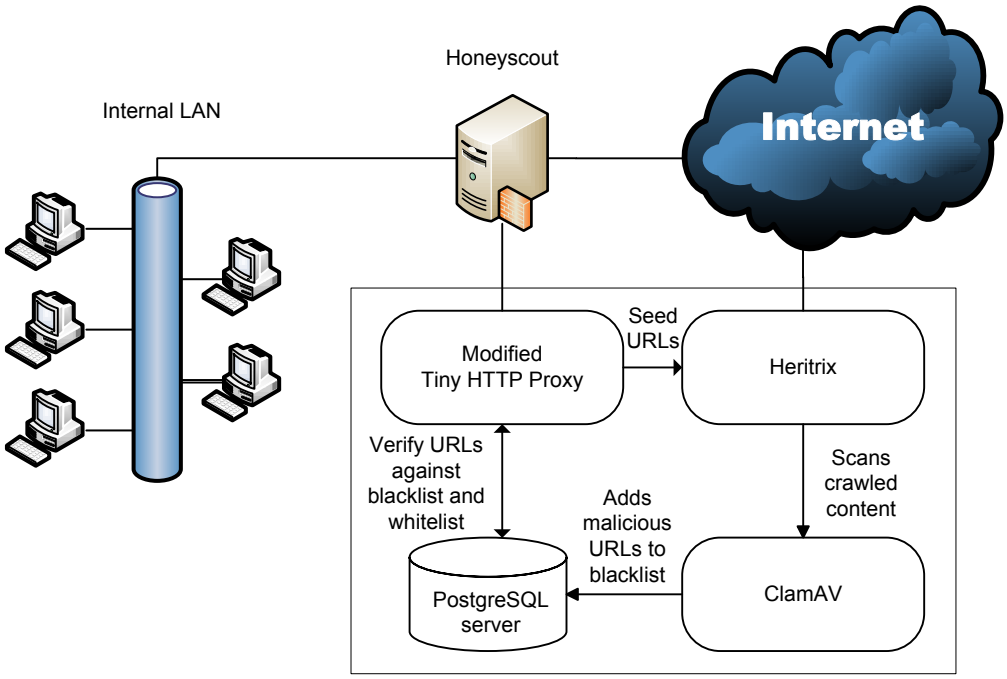

Figure 7: Illustration over Honeyscout's architecture.

4.3.1 List of major code changes

1. The Monkey-Spider scan script now deletes crawled content when scan is finished. It also adds found malware reports into the database in a slightly different way (se database modifications section 4.3.2 further down).
2. The mv_output and mw_scanner database tables has been dropped as they serve no prupose for Honeyscout.
3. Tiny HTTP Proxy has been modified to verify HTTP requests against the blacklist before letting it trough
4. Tiny HTTP Proxy also adds all requested URLs to a whitelist database (see section 4.3.3) if an entry for it don't already exist.
5. All HTTP requests intercepted by Tiny HTTP Proxy refering to browsable content and that does not exist in a whitelist entry is conveyed as seeds to Heritrix.
6. A honeyscout.py script has been written to make the process automatic. It starts the proxy, add new Heritrix crawl jobs via the jmxclient and continously scans finished crawls.
7. If a blacklisted URL is requested the honeyscout.py script will not forward the request but instead generate a custom page stating that the URL is blocked.

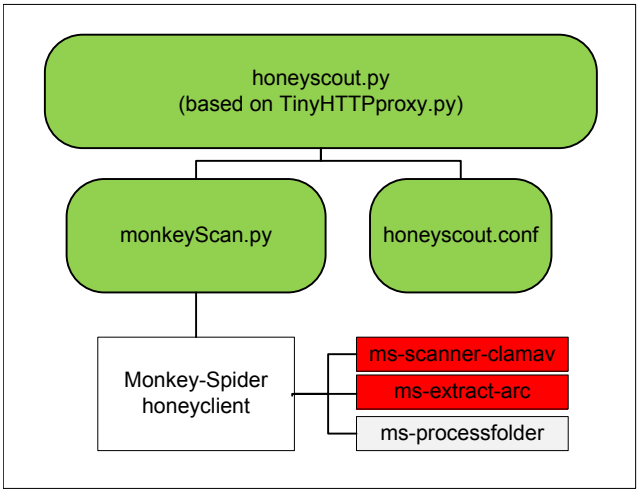

Figure 8: Honeyscout's file structure.
Green means new file written during the project, red means modified file and grey means unmodified file.

4.3.2 Blacklist database table

To better suit the purpose of Honeyscout the original Monkey-Spider malware database table has been modified to act as the blacklist. The table now looks according to figure 9. Because saving the malware for further examination is no longer interesting the *filename* and *size* fields from the original table has been dropped. The *comment* field containing output from ClamAV has been renamed to *reason* as it is more in-line with how the field now is used (i.e. the reason field is used to state why the URL is blacklisted). The checksum is an SHA1-hash of the malicious file and kept for reference.

```
id |          url          | checksum |        date         |        reason
---+-----------------------+----------+---------------------+---------------------
 1 | http://.../eicar.zip  | 3SK...GB | 2009-04-23 12:24:47 | Eicar-Test-Signature
 2 | https://www.finjan... | SN4...M7 | 2009-04-23 12:28:21 | VBS.Psyme-8
```

Figure 9: Excerpt from blacklist database table (shortened to fit document area).

4.3.3 Whitelist database table

To avoid crawling and scanning content several times when users on the network browse the same websites a whitelist database table was created. As seen in figure 10 the table contains address and date for all URLs scanned by Monkey-Spider and found clean from malware. As long as a URL exists in the whitelist requests to that specific URL will be automatically accepted and not sent to Heritrix as a seed. The date the URL was added to the whitelist is important because entries should not be permanently trusted after a successful scan. Therefore the whitelist has to be cleaned of entries that are too old. This clean up interval can be set depending on how often one wants to rescan websites that are generally trusted but could one day be hacked to spread malicious code.

url	date
http://www.postgresql.jp/document/pg702doc/user/x1133.htm	2009-04-23
http://isc.sans.org/	2009-04-23
http://www.sans.org/reading_room/	2009-04-23
http://www.svd.se/	2009-04-23

Figure 10: Excerpt from whitelist database table.

4.3.4 Configuration file

There are some variables that need to be set before Honeyscout can start running. To make it easy to configure Honeyscout when installed on a new system these variables are defined in a separate configuration file called *honeyscout.conf*. The configuration file are then processed when Honeyscout starts. The following variables need to be set:

doctypes - This defines the type of documents to cralw. Honeyscout will always crawl URLs that has no defined document type (URLs that ends with a /, such as in http//www.example.com/news/). The doctype config variable is an array that should contain the file endings of crawlable content. Its purpose is to avoid trying to crawl .txt, .avi or .doc files for example that would only produce errors in Heritrix. By default the following crawlable document types are set: *.html, .htm, .php, .asp, .aspx*.

jmxbin - The full path to the Heritrix JMX client binary.

login - Credentials to the Heritrix engine. The user should always be set to *controlRole* as this is the user allowed to start new crawl jobs via the JMX interface. Password is the administrator password set in the Heritrix configuration.

addr - Internet address (IP number) and port number to the Heritrix JMX interface. The IP address may be set to localhost if Heritrix is running on the same system as Honeyscout.

crawlerID - This variable is needed by the JMX client to find the right Heritrix crawler instance. Normally just one instance should exist and the only value that heeds to be changed is the *host* attribute and possibly the port numbers if anything else that the default ones are used. The same identifier string is displayed at the bottom of the Heritrix status page (see figure 11).

profile - Name of the Heritrix profile one desire to use for the crawls. A profile must first be created via Heritrix's web interface.

4.4 User interface

The user interface (UI) has not been a priority in the development of Honeyscout. However, there are definitely a need to somehow get information on the progress and state of the process. This sub-section presents the graphical feedback's available in Honeyscout.

4.4.1 Heritrix

The stand alone web crawl application Heritrix is configured and controlled solely from the web interface. Honeyscout makes use of JMX (see section 4.2.2), thus there is no need to use web UI except if one would like to create a new profile. Heritrix web interface does however provide a status page where the progress of current jobs as well as finished and pending ones can be viewed. This page can be interesting to view when using Honeyscout as it provides statistics and estimations as seen in figure 11. The web interface is accessed through a web browser on port 8080 and only accessible by localhost by default, but this can be changed in the configuration.

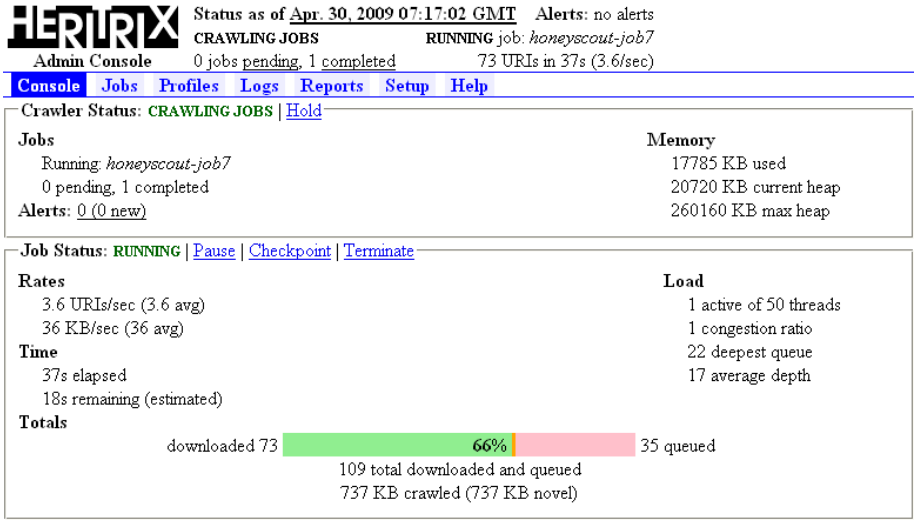

Figure 11: Heritrix's status page.

4.4.2 Honeyscout engine feedback

There is no graphical user interface available for Honeyscout but informative messages are written to the console. The type of messages given includes URLs requested, crawl jobs sent to Heritrix and started ClamAV scans. Figure 12 shows a screenshot of how a running instance of Honeyscout may look like.

```
chris@tom:~/proxy$ ./honeyscout.py
Any clients will be served...
Serving HTTP on 0.0.0.0 port 8000 ...
Starting to crawl http://www.remotedebugger.com/
05/06/2009 15:55:05 +0200 org.archive.jmx.Client addJobBasedon=test3,fb630
Scanning d2690a877cff7b630fa6baec7d828bbf-20090506134720545
URL whitelisted: http://www.rootsecure.net/
Scanning fb630deeb37c11ec98b550922c7391b1-20090506135504326
Starting to crawl http://www.milwOrm.com/
05/06/2009 15:57:03 +0200 org.archive.jmx.Client addJobBasedon=test3,5c042
```

Figure 12: Screenshot of Honeyscout's status messages.

Finished jobs are saved with a name represented by the md5-sum of the seed URL and timestamp, as can be seen in the *scanning ...* messages. This is to make sure the names are unique. The long lines cut off in figure 12 are output from the JMX client when starting a new job which proved hard to filter out. Originally the Tiny HTTP Proxy script displayed a lot of more information

about what connections where established and if they timed out. Such messages are not important in the context of Honeyscout's operation but the script could be extended to log helpful error messages either directly to the screen or to a log file.

4.4.3 Blocked pages

Users that try to access a URL that has been blacklisted needs to somehow get information on why the page cannot be viewed. Instead of just aborting the request leaving the user with a timed out connection attempt and a blank page Honeyscout generates a custom information page. The page, shown in figure 13, contains information that the requested URL has been blocked, the date it was added to the blacklist and the reason. Stated reason is the same information ClamAV adds to the blacklist database when malware is found. At this point the page generated is very basic, there is however big potential to extend functionality to allow user interaction with Honeyscout through this page. Users hitting a blocked page could for example be presented with an option to rescan the site for re-evaluation and possible whitelisting.

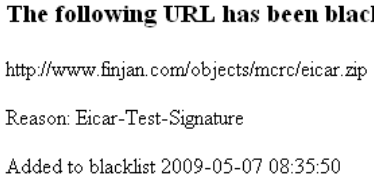

Figure 13: Screenshot of blocked page screen.

Since Honeyscout blocks only the specific malicious URL and not the whole domain websites that for example has been compromised to serve malware through an iframe (a frame inside a web page that can show content from another web page) the main page can still be accessed. Figure 14 shows an example where the site in question has an iframe normally showing ads. The URL pointing at the ads has however been blocked by Honeyscout and instead shows the blocked URL information page. All other clean content on the site is still accessible.

Figure 14: Screenshot of blocked URL inside an iframe

4.5 Limitations

Monkey-Spider is a low-interaction honeyclient which means that Honeyscout will be severely limited compared to the ideal case honeypot discussed earlier. The malware detection is signature based and thus will not find unknown malware. The gateway that monitors traffic acts as a HTTP proxy only and therefore only intercepts web browsing. Furthermore the users ability to interact with the software is limited due to programming and time constraints. It does however function as intended and will give good indications on how the concept works in practice. Heritrix also has some constraints when crawling websites, password protected pages for example, cannot be accessed and archived.

5 Evaluation

This section will try to evaluate the honeyscout concept. By using the proof-of-concept code that has been written for this thesis in a test environment connected to the Internet some answers, or at least pointers in the right direction, to the following questions are sought. What crawl scope works best? How much resources does a honeyscout need? Will the users notice anything else than the blocked pages?

In relation to the last question one thought abandoned early was that any site that was on neither the blacklist nor the whitelist would be inaccessible too. Users visiting a URL not already crawled and scanned would simply have to wait for the first scan to finish. This makes sense because allowing access to unscanned pages that could contain malware would be a security risk. The crawl and scan process, however, turned out to take quite a lot more time than expected. Users would not accept having to wait several minutes before a new URL could be accessed. The compromise came to be to still allow access to new URLs but keep the crawl short. This way users won't notice the Honeyscout at all unless they hit a blacklisted URL. Still the scan hopefully finishes quickly enough to catch any malware linked to from the main page before the user starts to follow links further. The compromise is better than it may seem. First of all malware is many times linked to from trusted sites (for example through Google ad text links [46]). Secondly the information that a malicious link has been accessed by a client in the network and that this client might be infected is still much better than not knowing about it at all. The problem could of course be marginalized by using a caching proxy that crawls content continuously and sends files independently to the honeyclient. This would make it possible to only allow access to verified content as well as speed up loading times for users in many cases. Using a caching proxy is discussed further in *future work* (chapter 6).

To evaluate the system a final test that involved crawl of 100 webpages was executed. 50 of those pages where known malware infected URLs taken from *malwaredomainlist.com*. Speed, malware detection and usability where the observed factors.

5.1 Crawl scope

There is a huge amount of possible crawl scope module combinations in Heritrix. After some initial tests two possible configurations have been chosen for further evaluation. One of the first ideas was to crawl whole domains (i e www.example.com) and whitelist/blacklist the entire site which would keep the number of database entries low. It however quickly became clear after the first couple of test crawls that crawling a whole domain would take too much time. Sites today contain lot of content and the crawl process isn't optimized enough for it to be practical. First of the two scopes chosen are *page crawl*, this crawl which is a combination of the default crawl setting in Heritrix and a depth limit of one. This means that all content on the seeded page will be crawled including all pages that are linked from it. The other is *path crawl*, which really is the path scope described in section 4.2.2 but build using the deciding scope and included filters. Using this setting would allow for whitelisting of whole website folders.

The final test used the *page crawl* settings. Almost solely because the overall

time it took to crawl content. In addition to the settings for page crawl already described some small changes to the Heritrix engine configuration was changed to increase crawl speed. These changes where a max accepted delay of 2000 ms, a maximum of 2 retries and a retry delay of 2 seconds. It should also be noted that Heritrix is hardcoded to be gentle on web servers and not send several request at once to the same sever. It is therefore not possible to get the theoretical best performance using Heritrix as a crawler.

5.2 Test enviroment

Equipment used during the tests are a 800 MHz VIA-chipset mini server with 1Gb of RAM. Not very powerful but it was the best alternative available much because it has three network interfaces and is therefore suitable to use as a proxy. All components of Honeyscout are run on this system which is loaded with a fresh install of Debian Linux. Network connection to the Internet is an unshared 100 Mbps line. Only one client computer is used to test the system but traffic from several computers is simulated by quickly visiting many different webpages at once.

5.3 Results

5.3.1 Speed

Speed turned out to be a bigger problem than anticipated. There are several different factors affecting the crawl speed such as the size of content on the page, the amount of links a page contains and webserver load. It is therefore hard to say what factor contributed to the slow speed most. The underpowered test system may also have been a big factor because speed results where a quite bit lower than the results achieved by Ali Ikinci during his thesis [12] using the same software. In his evaluation however several different sites were crawled in parallel which could explain the better performance. Average crawl speed of commercial and non-malicious sites was 2.1 URLs/sec or 34 Kb/sec, but these values ranged between 0-9.8 URLs/sec and 0-80 Kb/sec. Sites known to host malware where quite a bit slower. For reference a typical online news site's main page (including all the links) took about 15 minutes to crawl. Adding to this is the time it took for ClamAV to scan downloaded content. Analysis of content occurred with an average speed of 0.46 Mb/sec this was an expected value and inline with the results from the Monkey-Spider thesis.

5.3.2 Malware detection

The ClamAV engine which was fully updated with the newest virus definitions successfully detected 80 percent of the known malicious URLs taken from *malwaredomainlist.com*. Only valid URLs that still contained content where chosen for the test but it is hard to say if malicious code still existed at the URLs that went undetected by ClamAV.

5.3.3 Usability

As a user one does not notice Honeyscout more than a slight delay when visiting a new page, this is probably mostly due to the underpowered test system. The blocked URL page worked as intended but two unexpected errors occurred randomly which demanded a restart of the Honeyscout script.

6 Future work

The next step in what this thesis has defined as honeyscouts is definitely to use a high-interaction honeyclient as base. A good alternative would probably be Capture-HPC, but some of the commercial solutions, such as CWSandbox also look promising. Optimizing crawler behavior would also play an important role in improving efficiency. If the crawler could better determine what links should be followed first and continuously send data to the honeyclient for analysis, the concept would work much better. Because the biggest problem encountered during the practical tests done was the amount of time required to crawl and scan content. A more intelligent crawler together with a honeyclient that could receive and analyze a continuous flow of data would eliminate this user acceptance obstacle. A big improvement to crawl speed would also be to incorporate multiple crawler instances which can be done with The Heritrix Cluster Controller (hcc) [47]. Other improvements include better communication between the proxy and crawler to avoid multiple crawls of the same content. The whitelist implementation used in the project is supposed to minimize this problem but it does not work perfectly. Furthermore the usage could be extended beyond web traffic allowing in principle all data originating from the Internet to be analyzed by the honeyclient. At least e-mail traffic with attachments should be included as many attacks, especially targeted ones, use this delivery method.

Mentioned before was the integration with a caching proxy. First of all this could improve the general web browsing for users since content pre-crawled by the honeyscout would be cached and ready to be served if users later decides to request it. Another benefit is that the resources utilized by the honeyscout crawls are not wasted solely on malware detection as generated traffic is reused.

The following list summarizes the more interesting improvements that could further be considered to improve the honeyscout concept:

1. Integration with other honeyclients: preferably a high-interaction one.
2. More flexible options: optimized crawl techniques, intelligent proxy behaviour and more user-interaction.
3. Extended usage: scanning of all untrusted content, especially e-mails.
4. Integration with a caching proxy: to raise performance and save resources.

7 Conclusions

Using honeyclients to analyze content with the purpose to protect clients on the internal network does seem to be a good idea. Honeyclients, especially the high-interaction ones can be very effective in finding new malware that traditional antivirus products can't find. The honeyclients available today are however not ready for this kind of commercial use. Most of them are developed for research purposes and are not easily configured or quick enough to be used in a security product. This means that a lot of work has to be done before a commercially accepted product emerges.

Judging from the evolution of malware, new defensive technologies are definitely needed and a fully behavioral analysis of content is certainly effective and a logical next step to take. Performance and resource requirements do however question if the final and successful pro-active malware solution will utilize honeyclients in a conceptual way like the one studied in this thesis. Further studies should nonetheless prove to be interesting and, if the technology becomes more optimized, take place as a useful element in client computer defense strategy.

The work done in this thesis has somewhat found answers the questions asked in the introduction. Does it add something new to client security? Yes, the technology certainly has its place as a defense mechanism in active client malware protection. Will it work in a live production environment? It is very probable that the concept can work in a live production environment. There is however a lot of work and optimization that needs to be done. This thesis cannot give any definite answers to how and when such a solution is ready, only point out where most of the effort should be concentrated. Can it be made user-friendly? Yes, it is easy to get information to the user about the software behavior by replacing blocked pages with custom information pages. It is also easy to add tools on those same pages that allow users to interact with the software. How much resources are needed? Hard to specify but probably quite a bit, especially if high-interaction honeyclients are used. Yet there is room for efficiency improvements that could yield impressive results.

References

[1] Bruce Schneier. Are you sophisticated enough to recognize an internet scam? *The Mercury News*, 2003. http://www.schneier.com/essay-035.html.

[2] Bruce Schneier. Bruce Schneier on IT Insecurity. *CIO Insight*, 2008. http://www.schneier.com/news-073.html.

[3] Tim Greene. Client side attacks on the rise, SANS says. *NetworkWorld*, 2007. http://www.networkworld.com/news/2007/112807-client-side-attacks-rise.html?page=1.

[4] SANS Top-20 2007 Security Risks, 2007. https://www2.sans.org/top20/.

[5] F-secure. F-Secure IT Security Threat Summary for the Second Half of 2008, 2008. http://www.f-secure.com.

[6] Avira. Avira's forecast of malware trends for 2009, 2008. http://www.avira.com/en/company_news/malware_trends_for_2009.html.

[7] George Kurtz Stuart McClure, Joel Scambray. *Hacking Exposed 5th edition*. McGraw Hill, 2005.

[8] Jean-Marc Robert Thomas Chen. The Evolution of Viruses and Worms. Technical report, Dept. of Electrical Engineering, 2004. http://vx.netlux.org/lib/atc01.html.

[9] Analysis of the new "Code Red II" Variant. http://www.unixwiz.net/techtips/CodeRedII.html.

[10] Bruce Schneier. The Storm Worm. http://www.schneier.com/blog/archives/2007/10/the_storm_worm.html.

[11] Brett Stone-Gross et al. Your botnet is my botnet: Analysis of a botnet takeover. Technical report, University of California, 2009.

[12] Ali Ikinci. Monkey-spider: Detecting malicious web sites. Master's thesis, University of Mannheim, 2007.

[13] Gator information center. http://www.pcpitstop.com/gator/default.asp.

[14] AJP. Cookie Stealing Upgrade: Ajax Style. http://www.milw0rm.com/papers/130.

[15] XSS Shell website. http://www.securiteam.com/tools/6X00120HFO.html.

[16] Damien Cave. The parasite economy. *Salon.com*, 2001.

[17] Shashank Gonchigar. ANI vulnerability: History repeats. Technical report, SANS, 2007. http://www.sans.org/reading_room/whitepapers/threats/ani_vulnerability_history_repeats_1926.

[18] Steve Ragan. Adobe issues advisory on new PDF vulnerability. *The Tech Herald*, 2009.

[19] Stefan Pettersson. "visualizing endpoint security technologies using attack trees". Master's thesis, Linköping University, 2008.

[20] Paul C. van Oorschot Mohammad Mannan. On Instant Messaging Worms, Analysis and Countermeasures. Technical report, School of Computer Science, Carleton University, Ottawa, 2005. http://www1.cs.columbia.edu/ angelos/worm05/imworms.pdf.

[21] Ross Anderson Shishir Nagaraja. The snooping dragon: social-malware surveillance of the Tibetan movement. Technical report, University of Cambridge, 2009.

[22] Uli Ries. Steer clear of JavaScript packers, 2009. http://www.h-online.com/security/Steer-clear-of-JavaScript-packers-/news/113151.

[23] Thierry Zoller. Avast! - Generic evasion, 2009. http://blog.zoller.lu/2009/04/release-mode-forced-release-vendor-has.html.

[24] Christian Seifert et al. Know Your Enemy: Malicious Web Servers. Technical report, The Honeynet Project, 2007.

[25] FE Malware Research. The case against URL blacklists, 2008. http://blog.fireeye.com/research/2008/11/the-case-against-url-blacklists.html.

[26] Bruce Schneier. Is User Education Working, 2006. http://www.schneier.com/essay-139.html.

[27] Lance Spitzner. Definitions and Value of Honeypots. Technical report, The Honeynet project, 2003. http://www.spitzner.net/honeypots.html.

[28] Jonathan Rose. Turning the tables: Loadable Kernel Module Rootkits deployed in a honeypot environment. Technical report, SANS Institute, 2003.

[29] Free Software Foundation. GNU General Public License, version 2, 1991. http://www.gnu.org/licenses/gpl-2.0.html.

[30] Free Software Foundation. GNU General Public License, version 3, 2007. http://www.gnu.org/licenses/gpl-2.0.html.

[31] Client honeypot Wikipedia article. http://en.wikipedia.org/wiki/Honeyclient.

[32] The Honeynet Project, Capture-HPC website. https://projects.honeynet.org/capture-hpc/.

[33] VMware Server webpage. http://www.vmware.com/products/server/.

[34] The Honeynet Project, Capture-BAT website. https://www.honeynet.org/node/315.

[35] Christian Seifert. Capture-HPC mailinglist post: 3.0 Release?, 2009. http://public.honeynet.org/pipermail/capture-hpc/2009-March/000739.html.

[36] The Monkey-Spider project website. http://monkeyspider.sourceforge.net/.

[37] Heritrix website. http://crawler.archive.org/.

[38] ClamAV website. http://www.clamav.net/.

[39] Python website. http://www.python.org/.

[40] The Internet Archive. http://www.archive.org/.

[41] Capture-HPC mailinglist post by Nicolas, Thu 08 Jan 2009. http://www.mail-archive.com/capture-hpc@public.honeynet.org/msg00619.html.

[42] CWSandbox website. http://www.cwsandbox.org/.

[43] I. Ranitovic K. Sigurdsson, M. Stack. Heritrix user manual, 2009. http://crawler.archive.org/articles/user_manual/config.html.

[44] PostgreSQL community. PostgreSQL FAQ, 2009. http://wiki.postgresql.org/wiki/FAQ.

[45] Hisao Suzuki. Tiny HTTP Proxy website, 2006. http://www.okisoft.co.jp/esc/python/proxy/.

[46] Jose Vilches. Google sells text ads to known malware sites, 2008. http://www.techspot.com/news/32477-google-sells-text-ads-to-known-malware-sites.html.

[47] The Internet Archive. The Heritrix Cluster Controller, 2009. http://crawler.archive.org/hcc/.

A honeyscout.py

```
#!/bin/sh -
"exec" "python" "-O" "$0" "$@"
#-------------------------------------------------------------------------
# honeyscout.py
# Created by Christian Clementson May 2009
# Based on TinyHTTPProxy.py
#-------------------------------------------------------------------------


__doc__ = """Tiny HTTP Proxy.

This module implements GET, HEAD, POST, PUT and DELETE methods
on BaseHTTPServer, and behaves as an HTTP proxy.  The CONNECT
method is also implemented experimentally, but has not been
tested yet.

Any help will be greatly appreciated.   SUZUKI Hisao
"""

__version__ = "0.2.1"

import BaseHTTPServer, select, socket, SocketServer, urlparse, pg, cStringIO,
datetime, os, subprocess, md5, sys, glob, ConfigParser

class ProxyHandler (BaseHTTPServer.BaseHTTPRequestHandler):
    __base = BaseHTTPServer.BaseHTTPRequestHandler
    __base_handle = __base.handle

    server_version = "TinyHTTPProxy/" + __version__ + " Honeyscout mod"
    rbufsize = 0                        # self.rfile Be unbuffered

    def handle(self):
        (ip, port) =  self.client_address
        if hasattr(self, 'allowed_clients') and ip not in self.allowed_clients:
            self.raw_requestline = self.rfile.readline()
            if self.parse_request(): self.send_error(403)
        else:
            self.__base_handle()

    def _connect_to(self, netloc, soc):
        i = netloc.find(':')
  if i >= 0:
            host_port = netloc[:i], int(netloc[i+1:])
        else:
            host_port = netloc, 80
  #print "\t" "connect to %s:%d" % host_port
        try: soc.connect(host_port)
        except socket.error, arg:
            try: msg = arg[1]
            except: msg = arg
            self.send_error(404, msg)
            return 0
        return 1

    def do_CONNECT(self):
        soc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            if self._connect_to(self.path, soc):
                self.log_request(200)
                self.wfile.write(self.protocol_version +
                                 " 200 Connection established\r\n")
                self.wfile.write("Proxy-agent: %s\r\n" % self.version_string())
                self.wfile.write("\r\n")
                self._read_write(soc, 300)
        finally:
            #print "\t" "bye"
            soc.close()
            self.connection.close()

    def do_GET(self):
  scan = self.malware_detected()
  if scan[0]:
    self.genBlacklistedPage(scan[2],scan[1])
    return
        (scm, netloc, path, params, query, fragment) = urlparse.urlparse(
            self.path, 'http')
  if "." in path:
   type = path.split(".")
   if len(type) == 2:
    if type[1] in doctypes:
     seedURL(self.path)
   else:
    print "Error: fel antal punkter! " + path
  else:
   if self.path[-1] == "/":
    seedURL(self.path)

        if scm != 'http' or fragment or not netloc:
            self.send_error(400, "bad url %s" % self.path)
            return
        soc = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        try:
            if self._connect_to(netloc, soc):
                #self.log_request()
                soc.send("%s %s %s\r\n" % (
                    self.command,
                    urlparse.urlunparse(('', '', path, params, query, '')),
                    self.request_version))
                self.headers['Connection'] = 'close'
                del self.headers['Proxy-Connection']
                for key_val in self.headers.items():
                    soc.send("%s: %s\r\n" % key_val)
                soc.send("\r\n")
                self._read_write(soc)
```

```
        finally:
            #print "\t" "bye"
            soc.close()
            self.connection.close()

    def _read_write(self, soc, max_idling=20):
  iw = [self.connection, soc]
        ow = []
        count = 0
        while 1:
            count += 1
            (ins, _, exs) = select.select(iw, ow, iw, 3)
            if exs: break
            if ins:
                for i in ins:
                    if i is soc:
                        out = self.connection
                    else:
                        out = soc
                    data = i.recv(8192)
                    if data:
                        out.send(data)
                        count = 0
            else:
                print "\t" "idle", count
            if count == max_idling: break

    def malware_detected(self):
      q="SELECT reason,date FROM blacklist WHERE url = '%s'"%(self.path)
      s = db.query(q)
      if s.getresult() == []:
        return (False,'')
      else:
  print s.getresult()
        return (True,s.getresult()[0][0],s.getresult()[0][1])

    def genBlacklistedPage(self, date, reason):
     self.send_response(200)
     self.send_header("Content-type", "text/html")
     self.end_headers()

     # start HTML output
     self.wfile.write("<html><body>")
     self.wfile.write("<p><H3>The following URL has been blacklisted</H3></p>")
     self.wfile.write("<p>" + self.path + "</p>")
     self.wfile.write("<p>Reason: " + reason + "</p>")
     self.wfile.write("<p>Added to blacklist " + date + "</p>")

     # end HTML
     self.wfile.write("</html></body>")

    do_HEAD = do_GET
    do_POST = do_GET
    do_PUT  = do_GET
    do_DELETE=do_GET

class ThreadingHTTPServer (SocketServer.ThreadingMixIn,
                           BaseHTTPServer.HTTPServer): pass

def db_connect():
 try:
  dbc=pg.connect(dbname='exjobb',host='127.0.0.1',user='monkeyspider',
  passwd='kasspass1')
  return dbc
 except:
  print 'Unable to connect to database. Check your configuration.'
  exit(2)

def seedURL(url):
 date = datetime.date.today()
 q1 = "SELECT date FROM seeds WHERE url = '%s'"%(url)
 s = db.query(q1)

 if s.getresult() == []:
  addJob(url)
  q2="INSERT INTO seeds VALUES ('%s','%s')"%(url,date)
  db.query(q2)
  return
 else:
  print "URL whitelisted: "+ url
  return

def addJob(inputURL):
 seed = inputURL
 name = md5.new(seed)
 name = name.hexdigest()
 jobprofile = "addJobBasedon="+profile
 addjobcmd = jobprofile+","+name+",D,"+seed
 print "Starting to crawl " + seed
 job = subprocess.call("java " + "-jar " + jmxbin+" "+jmxlogin+" "+
 jmxaddr+" "+crawlerID+" "+ addjobcmd, shell=True)
 return


if __name__ == '__main__':
    db=db_connect()

    conf=ConfigParser.ConfigParser()
    conf.read("./honeyscout.conf")
    try:
      doctypes=conf.get('honeyscout','doctypes')
      jmxbin=conf.get('heritrix','jmxbin')
      jmxlogin=conf.get('heritrix','login')
      jmxaddr=conf.get('heritrix','addr')
      crawlerID=conf.get('heritrix','crawlerID')
      profile=conf.get('heritrix','profile')
    except:
      print 'Error in config file'
```

```
    exit()

job = os.spawnl(os.P_NOWAIT, "/home/chris/proxy/monkeyScan.py", "monkeyScan.py")
from sys import argv
if argv[1:] and argv[1] in ('-h', '--help'):
    print argv[0], "[port [allowed_client_name ...]]"
else:
    if argv[2:]:
        allowed = []
        for name in argv[2:]:
            client = socket.gethostbyname(name)
            allowed.append(client)
            print "Accept: %s (%s)" % (client, name)
        ProxyHandler.allowed_clients = allowed
        del argv[2:]
    else:
        print "Any clients will be served..."
    BaseHTTPServer.test(ProxyHandler, ThreadingHTTPServer)
```

B honeyscout.conf

```
# Configuration file for the Monkey-Spider system

[honeyscout]
doctypes = ["htm", "html", "php", "asp", "aspx"]

[heritrix]
jmxbin = "/home/chris/monkeyspider/heritrix-1.14.2/bin/cmdline-jmxclient-0.10.5.jar"
login = "controlRole:kasspass1"
addr = "10.0.0.1:8849"
crawlerID = "org.archive.crawler:guiport=8080,host=tom,jmxport=8849,
name=Heritrix,type=CrawlService"
profile = "test3"
```

C monkeyscan.py

```
#!/usr/bin/python
#-------------------------------------------------------------------------
# Created as a part of honeyscout.py by Christian Clementson May 2009
#-------------------------------------------------------------------------

import os, subprocess, sys, glob, time, ConfigParser

def finCheck(jmxbindir):
 heritrix_dir = jmxbindir[:-33]
 dirs = os.listdir(heritrix_dir+"/jobs/")
 for x in dirs:
         gzfiles = glob.glob(heritrix_dir+x+"/arcs/*.gz")
         if len(gzfiles) > 0:
          print "Scanning " + x
          subprocess.call(["ms-processfolder.py", heritrix_dir+"/jobs/"+x+"/arcs"])
          subprocess.call(["rm", "-R", heritrix_dir+"/jobs/"+x])
 return

conf=ConfigParser.ConfigParser()
    conf.read("./honeyscout.conf")
    try:
      jmxbin=conf.get('heritrix','jmxbin')
    except:
      print 'Error in config file'
      exit()

    while 1:
 time.sleep(20)
 finCheck(jmxbin)
 time.sleep(300)
```

D ms-scanner-clamav.py

```
#!/usr/bin/env python
#------------------------------------------------------------------------
# Modified for use with honeyscout.py by Christian Clementson May 2009
#------------------------------------------------------------------------
#$Id: ms-scanner-clamav.py 43 2009-03-24 07:34:44Z riker2000 $

# ms-scanner-clamav - Scans a given directory with clamav.
# Moves found malware to a seperate attic directory and updates
# the malware database.

# Copyright (C) 2006-2008 Ali Ikinci (ali at ikinci dot info)
#
# This file is part of the Monkey-Spider project.
# http://monkeyspider.sourceforge.net
#
# The Monkey-Spider project is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# the Monkey-Spider project is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with the Monkey-Spider project.  If not, see http://www.gnu.org/licenses/

# depends on clamav from http://www.clamav.net and pygresql from
# http://www.pygresql.org/

from os.path import basename
import sys

import ConfigParser
import os
import string

try:
    import pg
except:
    print 'Error importing the pg module. Check your pygresql installation'
    sys.exit(2)

def usage():
    print "Usage: ms-scanner-clamav.py [directory]"

def parseReportFile():

    config = ConfigParser.ConfigParser()
    config.read("/etc/monkey-spider.conf")
    try:
        dbName = config.get('ms-database', 'databasename')
        dbHost = config.get('ms-database', 'hostname')
        dbUser = config.get('ms-database', 'username')
        dbPass = config.get('ms-database', 'password')
        mwattic = config.get('ms-scanner', 'mwattic')
    except:
        print 'Unable to read configuration. Check the configuration file.'
        exit(2)

    #create attic for collected malware-binarys
    os.system("mkdir -p %s" % mwattic)

    #cdx file name
    cdxfile = basename(os.getcwd()) + ".cdx"

    f = open("../"+cdxfile, "r")
    cdx = f.readlines()
    f.close()

    # checksum index of the files for the reassosiaction of found malware
    cix = {}
    for x in range(len(cdx)):
        cix[x] = string.split(cdx[x])[5]

    f = open("clamav.report", "r")
    clamav = f.readlines()
    f.close()

    clamav_engine_version = string.split(string.split(clamav[0])[1], "/")[0]
    clamav_signature_version = string.split(string.split(clamav[0])[1], "/")[1]
    clamav_last_update = string.split(clamav[0], "/")[2][:-1]

    for i in range(2, len(clamav)):
        mw_name = string.split(clamav[i])[1]
        mw_filename = string.split(string.split(clamav[i])[0], ":")[0]
        mw_checksum = string.split((string.split(string.split(
string.split(clamav[i])[0], ":")[0], "/")[-1]), ".")[0]
        try:
            db = pg.connect(dbname=dbName, host=dbHost, user=dbUser,
  passwd=dbPass)
        except:
            print 'Unable to connect to database. Check your configuration.'
            exit(2)

        os.system("cp -u " + mw_filename + " %s" % mwattic)

        #Generate the next ids for databases malware and mw_scanner
        q = db.query("SELECT max(id) from blacklist")
        max_mw_id = q.getresult()[0][0]
        if max_mw_id == None:
            max_mw_id = 1
        else:
```

```
            max_mw_id = max_mw_id + 1

        id = max_mw_id
        filename = basename(mw_filename)
        checksum = string.split(basename(mw_filename), ".")[0]
        for x in range(len(cix)):
            if cix[x] == checksum:
                break
        url = string.split(cdx[x])[2].lower()
        size = 0
        date = string.split(cdx[x])[0]
        dateS = date[:4] + "-" + date[4:6] + "-" + date[6:8] + " " +
    date[8:10] + ":" +
    date[10:12] + ":" + date[12:14]
        q="INSERT INTO blacklist VALUES (%s,'%s','%s','%s','%s')"%(
    id,url,checksum,dateS,mw_name)
        db.query(q)
        db.close()

def main():

    if (len(sys.argv) != 2):
        usage()
        sys.exit(2)

    if not os.path.exists(sys.argv[1]):
        print "Error: Path does not exist or you don't have the needed permissions"
        sys.exit(2)

    if not os.path.isdir(sys.argv[1]):
        usage()
        sys.exit(2)

    #chdir to where the arc file resides
    workdir = sys.argv[1]
    os.chdir(workdir)

    print 'Scanning folder %s for viruses with ClamAV' % workdir,

    #Scan directory with clamav and generate report file
    os.system("clamdscan -V > clamav.report")
    os.system("clamdscan |grep FOUND >> clamav.report")

    parseReportFile()

    print '.'


if __name__ == "__main__":
    main()
```

E ms-extract-arc.py

```
#!/usr/bin/env python
#------------------------------------------------------------------------
# Modified for use with honeyscout.py by Christian Clementson May 2009
#------------------------------------------------------------------------
#$Id: ms-extract-arc.py 43 2009-03-24 07:34:44Z riker2000 $

# ms-extract-arc - Dump all files contained in an Internet Archive ARC File together
#                  with a cdx index file

# Copyright (C) 2006-2008 Ali Ikinci (ali at ikinci dot info)
#
# This file is part of the Monkey-Spider project.
# http://monkeyspider.sourceforge.net
#
# The Monkey-Spider project is free software: you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation, either version 3 of the License, or
# (at your option) any later version.
#
# the Monkey-Spider project is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
# GNU General Public License for more details.
#
# You should have received a copy of the GNU General Public License
# along with the Monkey-Spider project.  If not, see http://www.gnu.org/licenses/

# depends on the acreader tool from the Heritrix package
# (http://crawler.archive.org)

import sys

import mimetypes
import os
import re
import string

def stripHeader(filename):
    # strip the protocol header from the file contents
    file = open(filename, 'r')

    idx = 0

    isFirstLine = True
    while True:
        result = re.compile(":").search(file.readline())
        idx = file.tell()
        if result == None and isFirstLine == False:
            break
        isFirstLine = False

    file.seek(idx)
    outfile = open(filename[:-4], "w")
    outfile.write(file.read())
    outfile.close()

    file.close()
    os.remove(filename)

def usage():
    print 'Usage: ms-extract-arc.py [filename].arc.gz'

def main():

    if (len(sys.argv) != 2):
        usage()
        sys.exit(2)

    if not os.path.exists(sys.argv[1]):
        usage()
        print "Error: File does not exist or you don't have the needed permissions"
        sys.exit(2)

    if not os.path.isfile(sys.argv[1]):
        usage()
        sys.exit(2)

    if sys.argv[1][-7:] != ".arc.gz":
        usage()
        print "Error: File is not in *.arc.gz Format"
        sys.exit(2)


    arcfile = os.path.basename(sys.argv[1])

    # change workingdir to where the arc file resides
    workdir = os.path.dirname(os.path.abspath(sys.argv[1]))
    os.chdir(workdir)

    print 'Extracting %s' % arcfile,

    # create a directory where the arcfile will be extracted, remove an older
    # one if there is
    arcdir = arcfile[:-7]
    if os.path.exists(arcdir):
        os.system("rm -rf " + arcdir)
    os.system("mkdir " + arcdir)
    os.system("gunzip " + arcfile)

    # generate cdx file with the arcreader tool from the Heritrix package
    cdxfile = arcfile[:-6] + "cdx"
    arcreaderinput = os.path.abspath(arcfile[:-3])
    os.system("arcreader -d true  " + arcreaderinput + " > " + cdxfile + ".tmp")
    os.system("mv " + cdxfile + ".tmp " + cdxfile)
```

```
    # open the cdxfile
    f=open(cdxfile,"r")
    cdx = f.readlines()
    f.close()

    g = open(arcfile[:-3], "r")

    for x in range(2, len(cdx)):
        archiveThisFile = False
        cdxstring = string.split(cdx[x])

        # generate file name
        fext = mimetypes.guess_extension(cdxstring[3])
        if (fext == None):
            fext = ".raw"

        findex = cdxstring[5]
        fname = findex + fext

        # Seek to the right position
        g.seek(int(cdxstring[6]))
        g.readline()
        h = open(arcdir + "/" + fname + ".tmp", "w")
        h.write(g.read(int(cdxstring[7])))
        h.close()

        stripHeader(arcdir + "/" + fname + ".tmp")

    g.close()

    os.system("rm "+arcfile[:-3])
    print 'Done  extracting %s'%arcfile

if __name__ == "__main__":
    main()
```

Presentationsdatum

2009-06-12

Publiceringsdatum (elektronisk version)

2009-09-01

Institution och avdelning
Institutionen för systemteknik

Department of Electrical Engineering

Linköpings universitet

Språk

___ Svenska
_x_ Annat (ange nedan)

Engelska

Antal sidor
40

Typ av publikation

____Licentiatavhandling
_x__Examensarbete
____C-uppsats
____D-uppsats
____Rapport
____Annat (ange nedan)

ISBN (licentiatavhandling)

ISRN LiTH-ISY-EX--09/4262--SE

Serietitel (licentiatavhandling)

Serienummer/ISSN (licentiatavhandling)

URL för elektronisk version
http://www.ep.liu.se

Publikationens title
Client-side threats and a honeyclient-based defense mechanism, Honeyscout
Hot på klientsidan och en försvarsmekanism baserad på klient honungsfällor; Honeyscout

Författare
Christian Clementson

Sammanfattning

Abstract
Client-side computers connected to the Internet today are exposed to a lot malicious activity. Browsing the web can easily result in malware infection even if the user only visits well known and trusted sites. Attackers use website vulnerabilities and ad-networks to expose their malicious code to a large user base. The continuing trend of the attackers seems to be botnet construction that collects large amounts of data which could be a serious threat to company secrets and personal integrity. Meanwhile security researches are using a technology known as honeypots/honeyclients to find and analyze new malware. This thesis takes the concept of honeyclients and combines it with a proxy and database software to construct a new kind of real time defense mechanism usable in live environments. The concept is given the name Honeyscout and it analyzes any content before it reaches the user by using visited sites as a starting point for further crawling, blacklisting any malicious content found. A proof-of-concept honeyscout has been developed using the honeyclient Monkey-Spider by Ali Ikinci as a base. Results from the evaluation shows that the concept has potential as an effective and user-friendly defense technology. There are however large needs to further optimize and speed up the crawling process.

Nyckelord
Client security, network security, malware, virus, honeypot, honeyclient,capture-hpc, monkey-spider, honeyscout