Blockchain-based command and control for next generation botnets

Mengidis Anagnostis

SID: 3307160007

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Communications and Cybersecurity

DECEMBER 2018

THESSALONIKI – GREECE

Blockchain-based command and control for next generation botnets

Mengidis Anagnostis

SID: 3307160007

Supervisor:	Dr. Georgios Ioannou
Supervising Committee Members:	Assoc. Prof. Name Surname
	Assist. Prof. Name Surname

SCHOOL OF SCIENCE & TECHNOLOGY

A thesis submitted for the degree of

Master of Science (MSc) in Communications and Cybersecurity

DECEMBER 2018

THESSALONIKI – GREECE

Abstract

This dissertation was written as a part of the MSc in Communications and Cybersecurity at the International Hellenic University.

This work will try to address one of the main issues that modern botmasters face, which is the takedown of their Command and Control infrastructure. After establishing the scope of the problem, we will propose our solution.

Mengidis Anagnostis

7/12/2018

Contents

ABSTRACT ..... **III**

CONTENTS ..... **V**

1 INTRODUCTION **1**

1.1 BOTS AND BOTNETS 1

1.2 AIM OF THIS DISSERTATION 2

1.3 OBJECTIVES OF THIS DISSERTATION 2

2 FUNDAMENTALS ABOUT BOTNETS **5**

2.1 BOTNET HISTORY 5

2.2 BOTNET LIFE CYCLE 6

2.3 ARCHITECTURAL DESIGNS 8

2.3.1 Centralized C&C *8*

2.3.2 Decentralized C&C *9*

2.3.3 P2P C&C *10*

2.4 BOTMASTER CHALLENGES 12

2.4.1 Sinkholing *12*

2.4.2 P2P Polluting *13*

2.4.3 C&C Takedown *14*

3 BLOCKCHAIN TECHNOLOGY **15**

3.1 BLOCKCHAIN DEFINITION 15

3.2 HOW BITCOIN WORKS 16

3.2.1 Addresses and Transactions *17*

3.2.2 Block Propagation in Bitcoin Network *20*

3.3 HOW ZOMBIECOIN USED BITCOIN'S NETWORK 21

3.4 WHAT IS MONERO 21

3.4.1 Addresses and Transactions *22*

3.4.2 Disadvantages of Bitcoin as a C&C Communication Channel *26*

4 MONERO AS A SOLUTION 29

4.1 OPERATION OF OUR PROPOSED BOT 29

4.2 TESTING ENVIRONMENT 31

4.3 SENDING COMMANDS 32

4.4 PROOF OF CONCEPT 34

4.5 COST OF OPERATION 39

5 CONCLUSIONS 41

BIBLIOGRAPHY 43

1 Introduction

Over the last decades, the undeniable evolution of technology, has offered innumerable positive effects for the society. However, as our world becomes increasingly interconnected, the emergence of online threats is an unavoidable side effect which needs to be addressed.

Internet threats have become a hot topic of discussion during the last past years and despite their short-lived history, they are undergoing an impressive transformation from being a threat which solely aims to disrupt the regular function of infrastructure, to a threat that also targets organizations and people. This leads to an unprecedented versatility of threats makes the Internet an excellent medium for cyber criminals which in turn transforms the cyberspace into a de facto theater of war. It gives the opportunity of accessing billions of interconnected devices at almost real-time without having to rely on means of interference such as physical presence or physical access. Additionally, the anonymity that internet offers, allows bad actors to disguise their origin and their motives, which places them in a position to orchestrate large-scale attacks without worrying about facing the consequences or sometimes realized the impact of their attacks since they are so detached from their targets.

One of the most common goals of any cyber-attack is gaining access to a remote system, thus performing tasks that in any other case could only be possible for the system's user. Depending on the sophistication of the attack, it is possible for the attacker to even escalate that user's privileges and gain escalated permissions, equal to these of a system's administrator, which can be later used to steal sensitive data, disrupt normal operation and hinder those system's usual workflow.

1.1 Bots and Botnets

Instead of manually controlling compromised systems, bad actors commonly install malware on the victim's system so that they are able to autonomously perform actions based on predefined commands by the attacker. This approach mainly aims at increasing the

number of infected hosts, thus increasing cyber criminal's capabilities, by automating the process of spreading over networks, removable storage devices or even email attachments. The biggest drawback of this classic approach is that as soon as the malware starts spreading, the attacker can no longer interfere with it, update the payload in case a bug is found or make any changes in the malware's predefined list of malicious activities.

In order to avoid these shortcomings, malware authors came up with the idea of implementing communication channels that will allow them to communicate directly with the infected systems, therefore allowing them to issue commands on demand without having to rely on beforehand programmed script of actions. This generation of malware are called *Bots*, derived from word robot, the network of compromised hosts is called a *Botnet* and the operator who issues the commands and usually owns the botnet is called *Botmaster*.

1.2 Aim of this Dissertation

In this dissertation, we will try to present a novel idea of a blockchain-based Command and Control (C&C) mechanism that can be potentially very difficult to countermeasure. It is our belief that despite the numerous advantages of blockchain technology, it can also be a very powerful tool in the hands of malware authors that will allow them to develop the next generation of botnets. As far as we know, there has been only one work similar to this dissertation and our aim is to keep the main idea and propose an even more resilient C&C mechanism.

1.3 Objectives of this Dissertation

We will first establish a base on what are botnets, their typical lifecycle, the past and current mechanisms utilized by them, and what are the most common countermeasures against them.

In the third chapter we will analyze how blockchain works and present Zombiecoin, a botnet-related research that used Bitcoin's network for relaying commands. We will

then proceed to examine Monero as an alternative blockchain platform which mainly focuses in anonymity and privacy.

In the fourth chapter we will examine how realistic is to use Monero's blockchain as a communication channel for a botnet and create a proof of concept.

2 Fundamentals about Botnets

In this chapter we will attempt to demystify the way botnets work and examine the different methods used by malware authors to communicate with the attacker along with some of the most commonly used techniques utilized by them in order to avoid detection. Furthermore, we will try examining from a botmaster's perspective, the challenges that he has to face in order to avoid the takedown of his botnet.

2.1 Botnet History

Similar to their predecessors, namely viruses and worms, bots are using similar methods in order to self-propagate. These methods resemble a lot those of other classes of malware and typically include the exploitation of vulnerabilities in the software [1], trojan injection [2] and the use of social engineering [3].

Historically, botnets originated from IRC which stands for Internet Relay Chat and is a chat system based on text. The concept of the first bots was perceived due to the need to interpret simple commands given by the chat users and help chat rooms administrators retrieve information about emails and aliases. The first recorded IRC-based bot, released in 1993, was Eggdrop [4] and its development continues until today. Eggdrop has triggered the development of other bots but this time these bots were created for the primary purpose of attacking other IRC users or other servers. Consequently, bots began having new features implemented, such as Denial of Service (DOS) and Distributed Denial of Service shortly after [5].

That lead to the evolution of bots which began using complex communication mechanisms, integrate powerful attack methods and updating in a very fast pace, exploiting even zero-day vulnerabilities. Examples of such bots are SDBot [6] and Agobot [7]. The latter is considered by many as the turning point for the botnet ecosystem since at that point, no one considered botnets as a major threat to the Internet [8].

Eggdrop	1993	Centralized	IRC	None	[4]
GTbot	1998	Centralized	IRC	None	[9]
SDbot	2002	Centralized	IRC	PE Injection	[10]
Agobot	2002	Centralized	IRC	MS03-026 exploit	[11]
Kraken	2006	Centralized	UDP,TCP	Social Engineering	[12]
Rustock	2006	P2P	HTTP	Spam email	[13]
Storm	2006	P2P	UDP+ / eDonkey	Spam email and Social Engineering	[14]
Conficker	2008	Centralized	HTTP	MS08-067 exploit	[15]
Conficker C	2009	P2P	TCP, UDP	MS08-067 exploit, NETBIOS	[16]
Festi	2010	Centralized	HTTP	Spam email	[17]
TDL-4	2011	P2P	Kad Network	MS10-092 exploit	[18]
Zeus	2011	P2P	UDP, TCP, HTTP	Spam email	[19]

Table 1

The latest generation of botnets have become even more sophisticated and can now propagate through file sharing, P2P networks and drive-by downloads in websites. In Table 1, we list some of the more well-known bots along with their main characteristics.

2.2 Botnet Life Cycle

A typical life-cycle of a botnet has five phases which includes the initial infection, the secondary injection, establishing a connection, the C&C and finally the update and maintenance of the botnet. Literature sometimes uses different terms of all the different phases, but generally a typical botnet life cycle is as shown in Figure 1.

During the first phase, the Initial Infection, the host is infected and becomes a potential zombie which is another commonly used term for bots. In this phase, typical infection methods are used, like exploiting vulnerabilities in the operating system, infected downloads from webpages, infected email attachments or automatically executed scripts in USB removable devices [20, 21].

During the second phase, the payload is being injected to the host, usually downloaded by a list on network addresses, contained in the bot which infected the host during the initial stage. These addresses can vary from a list of IP addresses to a list of domain names. Even though this increases the resilience of the botnet to take down attempts, at

the same time it creates a single point of failure, something that will be examined later in this dissertation. As soon as the payload is downloaded and executed, the host starts behaving as a bot and becomes part of the botnet. The payload is usually downloaded through HTTP, FTP or P2P network protocols.

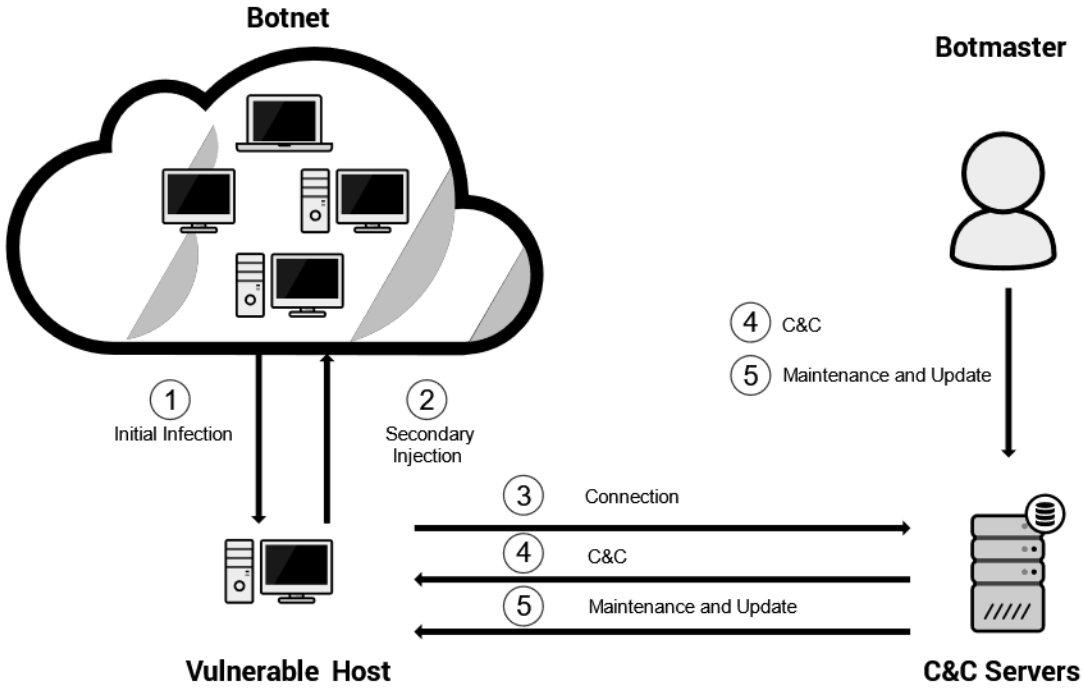

Figure 1

The third phase of the infection is probably the most critical one, since the initial connection to the C&C server is established in order to receive updates or new instructions. Some authors mention this phase as Rallying [22], however the process remains the same and is usually performed each time the host is restarted to ensure that has the latest version of the payload and report back to the C&C server that it is still part of the botnet and it is ready to receive new commands. Evidently, in a typical bot life cycle, this phase will be repeatedly executed numerous times [23]. Some researchers tend to merge the second and third phase into one because they are close related. There are many bots that use the C&C server as a file server for the payload too, hence the two phases are more likely to happen simultaneously.

The last phase is the update and maintenance of the botnet. Most botmasters have to update the payload in order to continue avoid detection by antivirus software, apply new features that the malware author implemented or even switch C&C server to obfuscate the footprint that their botnet leaves behind. Previous work has shown that the last change

can be observed by monitoring an increase in DNS queries performed by all hosts in a very short amount of time [24].

2.3 Architectural Designs

As mentioned previously, what distinguishes botnets from other types of malware, is the use of C&C channels which make possible for the botmaster to update and direct them. These channels can be operated through a variety of network topologies and use different transport layers such as TCP or UDP. A simplistic view of a botnet can be seen in Figure 2.

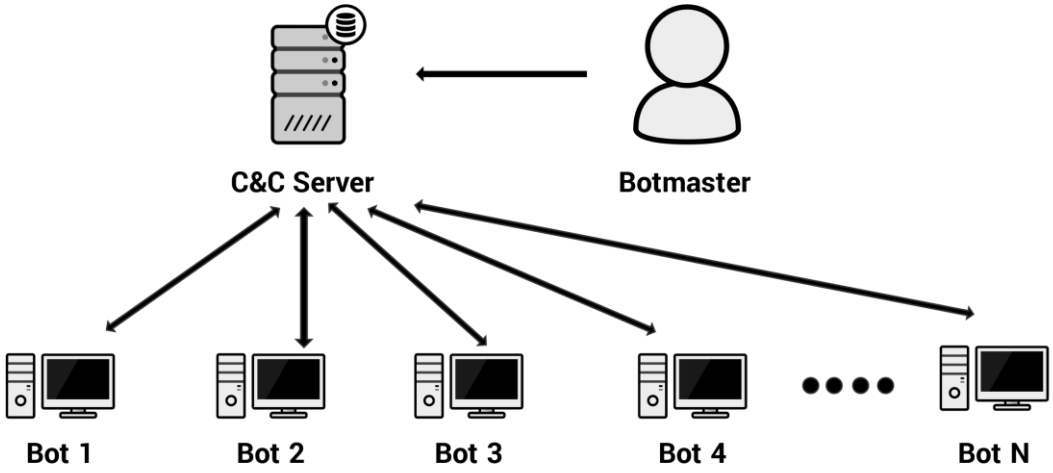

Figure 2

As seen in Table 1, depending on their C&C architecture, botnets can be classified as Centralized, Decentralized and P2P.

2.3.1 Centralized C&C

The centralized C&C architecture shares a lot of similarities with the classic server-client network model, as shown in Figure 3. Examples of such architecture are botnets that rely on IRC protocol due to IRC's server-centric nature and HTTP based.

In the case of IRC, the owner of the botnet has to create IRC channels on the C&C server where bots idle until a new command arrives. IRC protocol was very popular during the past, as indicated by the report issued by Symantec in 2010, which indicated that more than 30% of the botnets used IRC as their communication protocol.

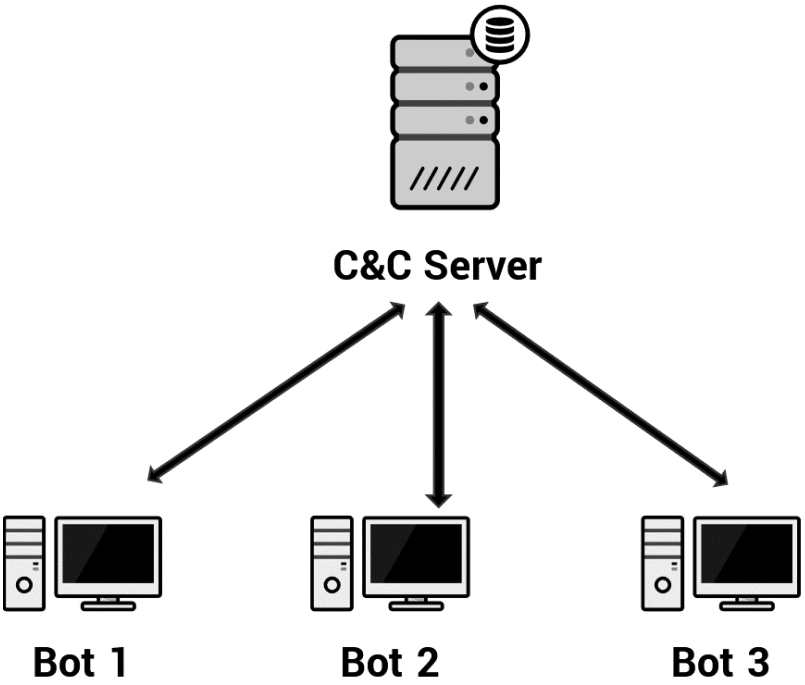

Figure 3

However, due to the reason that it is considered good practice for network administrators to block IRC traffic [25], the HTTP protocol became a more popular option for C&C communication. The biggest advantage that HTTP protocol offers is that ports 80 and 8080 are permitted in almost every network and the only for network administrators to distinguish botnet traffic from usual web traffic is by using IDS or some packet sniffer and perform traffic analysis.

The centralized kind of architecture offers low latency and easier coordination of the botnet in such a way that enables easier monitoring by the botmaster. Additionally, it is much simpler to maintain such a network and the deployment time of a single server is significantly smaller than the architectures presented below.

2.3.2 Decentralized C&C

Even though centralized C&C architecture offers great manageability due to its straightforward design, it is also the botnet's single point of failure. In case someone takes down or denying access to the C&C server will automatically render the botnet useless.

This weakness led to the development of the next generation botnets where they tried to maintain the efficiency of the centralized model as much as possible but improve its

resilience against takedowns. This was accomplished by adding redundant C&C servers which are contacted either sequentially or using round-robin [26]. This topology is shown in Figure 4.

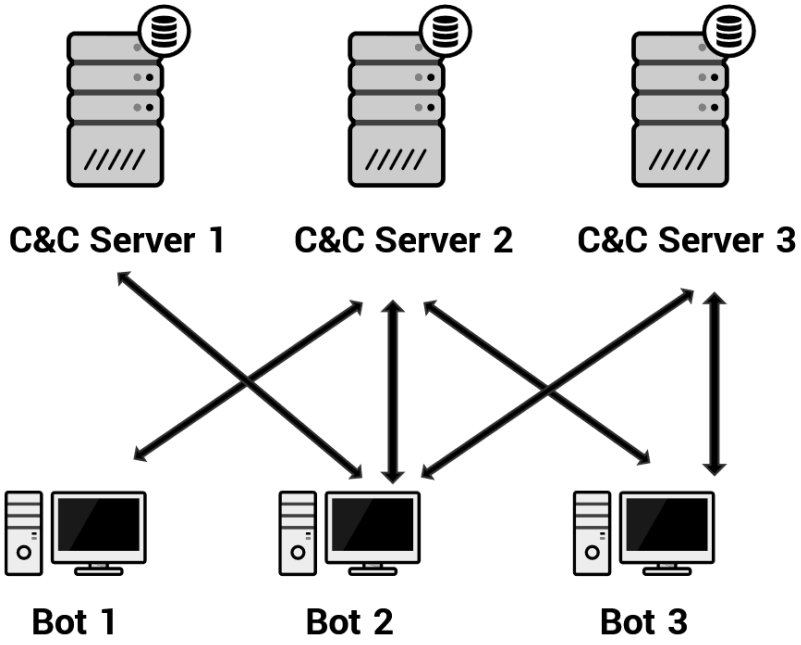

Figure 4

Some more advanced botnets are using Domain Registration Algorithms which allows the botmaster to register many domains ahead of time that will be used by the C&C servers in the future. However, since the algorithm that creates these domains is usually hard-coded in the payload, it is possible through reverse engineering to identify which domains are going to be registered. Consequently, a defender can register them before the botmaster and hijack the botnet.

2.3.3 P2P C&C

The most recent generation of botnets adopts a more random architecture in order to avoid the disarticulation of the network even when a C&C server is taken down. Such architecture is based on a variety of P2P protocols and a typical topology of such botnet is illustrated in Figure 5.

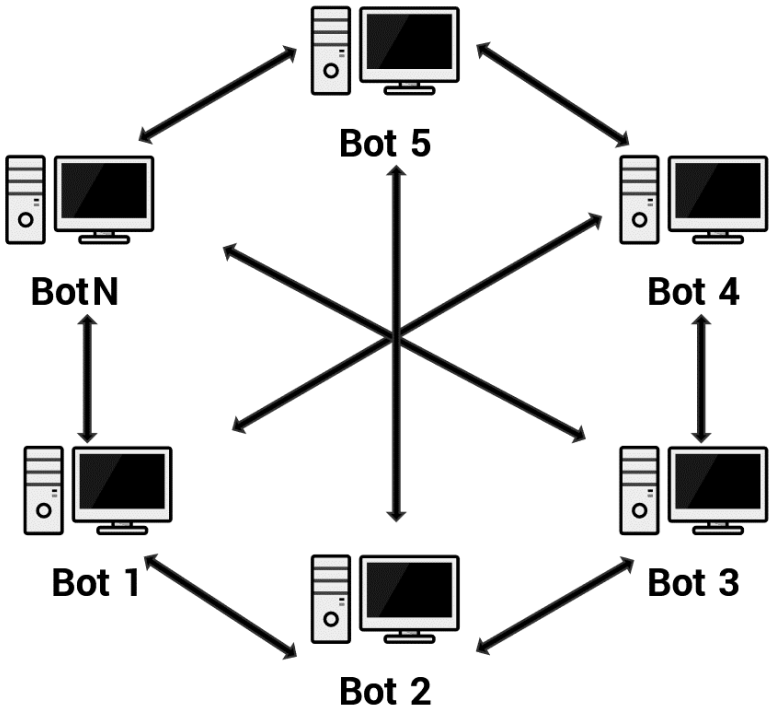

Figure 5

According to Jelacity and Bilicki [27], depending on the overlay used, they can be classified as follows:

- Unstructured P2P Overlays: This overlay network contains random topologies such as uniform random or power-law networks. They do not support features like routing or flooding.
- Superpeer Overlays: In this type of networks, some peers are randomly selected to play the role of temporary servers. These peers are called superpeers and are used my popular applications like Skype [28].
- Structured P2P Overlays: These networks create a mapping of the network based on a Distributed Hash Table (DHT). Most recent botnets use this type of structure and more specifically the Kademlia DHT [29].

In P2P botnets, after the initial infection takes place, bots are searching the network to find other peers to connect to, so that they integrate to the rest of the botnet. This procedure is called bootstrap and is considered the weakest point of such botnets, mainly because the discovery of the initial peer list compromises the rate at which the network can grow [30].

2.4 Botmaster Challenges

Even though preventing new hosts from being infected slows down the growth of an active botnet, the hosts that are already infected are not affected directly. Therefore, the bots are still operating regularly and perform the malicious activities that their botmaster instructs them to. To cope with these botnets, various techniques have been proposed, some of which have already proven effective in mitigating a botnet's operation.

2.4.1 Sinkholing

When it comes to mitigating botnets, one major challenge is to observe the communication of the infected machines in the first place. Since logging the communication of all infected machines is close to impossible, observing the communication of the C&C server is a more realistic approach. Nevertheless, this approach will not always be possible since most of the times the C&C server is beyond the scope of influence of the defending organization.

This problem can be alleviated through what is known as DNS-based sinkholing. This method can be applied to any botnet that uses fixed domain names, with or without fast fluxing, and basically takes advantage of the fact that every bot needs a DNS server to resolve the IP address of the C&C server that it tries to communicate with. In a controlled network like a corporate LAN, the local nameserver is configured is such a way that the domain of the actual C&C server will be redirected to the IP address of the sinkhole server, thus exposing the IP addresses of all the infected machines.

If the bot uses domain fluxing [31], sinkholing technique can also be applied to environments beyond the control of the defender or even the whole Internet. When using domain fluxing, each bot generates a list of domains that are possible candidates for the C&C server.

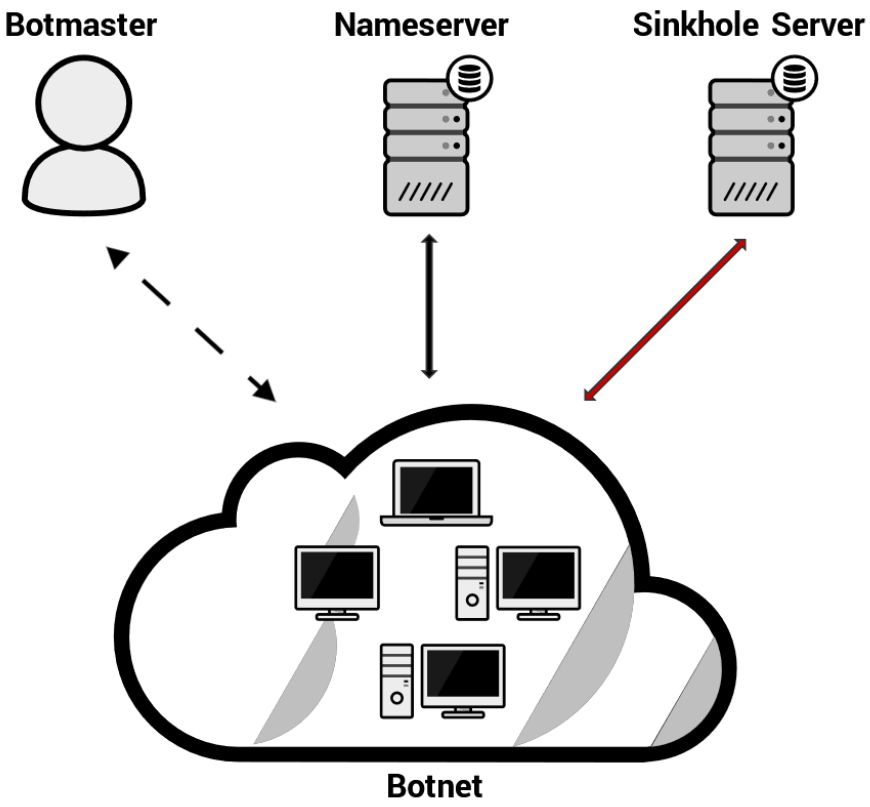

Figure 6

Usually these generated domain names are using a similar seed with respect to something like the current date or timestamp. In case a domain cannot be resolved, the infected host continues to the next generated domain until a C&C server responds. As a result, if someone reverse engineers a bot sample, he can generate the list of all possible domains, register them before the botmaster and redirect the traffic to the sinkhole server.

In the case of P2P botnets, since domain names are not used, we can apply the sybil-attack method. Based on a metric defined by the developer of the bot, each bot knows about a subnet of other bots, usually the ones with the smaller routing distance. This can be exploited by deploying a number of fake peers which will route all the messages intended for the C&C to the sinkhole server, as demonstrated by [14].

2.4.2 P2P Polluting

Aside from sybil-attacks, P2P based botnets are also susceptible to a technique called P2P polluting. A P2P network's peers do not know about every other single peer and only carry a limited list of them. This can be exploited by deploying a number of fake peers to the already existing peers of the botnet. Even though this sounds similar to the sybil-

attack, the main difference lies in the fact that in this case the peers are not real, and they are just used to fill the list of the actual ones.

By overwriting all the legit entries in the aforementioned list with fake peers, we manage to propagate all these fake entries to the rest of the botnet which slowly begins to fork from the rest of the network. Consequentially, if no bot is able to communicate with the other, no messages will be relayed, thus the connection with the C&C server will be lost.

Some malware authors however came with a countermeasure, where they try to probe the peers before adding them to the peer list. If the peer responds then and only then it is added to the list, otherwise it is being discarded as a fake one.

2.4.3 C&C Takedown

The most classic countermeasure towards existing botnets, is the takedown of their C&C server. If someone analyzes a malware sample, it is possible to retrieve the address of the server that plays the role of the C&C. Through IP lookup or reverse lookup, the organization that owns the specific IP can be found and the server can be disconnected or shut down.

Depending on the legislation that applies to the organization, the law enforcement agencies can issue a demand to the Internet Service Provider to cease the connectivity to the specific server. As soon as the C&C server loses its connectivity to the internet, the botmasters can no longer issue new commands, therefore the botnet will continue following the instructions that were already given to it but since it will no longer be able to update, it will eventually remain dormant. Such an example is the takedown of the Rustock botnet which was a combined effort by Microsoft Digital Crimes Unit, the U.S. Marshals Service and the U.S. District Court [32].

Even though these takedowns sound relatively easy for any organization located in a developed nation, the use of bulletproof hosting [33] by the botmasters makes the takedown a much harder procedure.

3 Blockchain Technology

Cryptocurrencies became popular due to the surprising and quick rise of Bitcoin which showed an unprecedented growth during the last couple of years. As of January 2017, more than 1000 cryptocurrencies are in circulation even though most of them are either abandoned by their developers or their trading volume is practically non-existent. Arguably however, the biggest innovation that cryptocurrencies introduced was the utilization of blockchain as a distributed database.

3.1 Blockchain Definition

The idea of cryptocurrencies was first perceived by David Chaum in his proposal for untraceable payments [34] where he described a system where third-parties are unable to determine payees and time or amount of payments made by an individual. He took his idea one step further in 1990 by creating the first cryptographic anonymous electronic cash system, known as ecash [35]. Later in 90s, a lot of startups emerged trying to implement electronic cash protocols, attempts that ultimately failed.

Cryptocurrencies, as we know them today, are peer-to-peer decentralized digital assets based on the principles of cryptography. Most cryptocurrencies use a distributed database as the pillar of their system, known as Blockchain, which allows them to use it as a distributed public ledger without having to rely to any form of centralized control similar to banking systems.

The blockchain is the equivalent of a book maintained by a bank which contains all the accounts and each transaction made. Of course, this is an oversimplification and in reality, there are many differences, possibly the most noticeable being the fact the bank's records are private whereas the blockchain is publicly available and easily accessible by everyone. One of the most interesting aspects of blockchains is that they contain the records of every transaction made since the beginning, also known as genesis block, by using

a peer-to-peer distributed timestamp server which generates computational proof of the chronological order of the transactions [36]

3.2 How Bitcoin Works

Bitcoin (BTC) is the first widely used cryptocurrency and as of today, remains the most widely adopted one. It was created in 2008, when pseudonymous Satoshi Nakamoto posted a paper [36], describing a system for trustless electronic transactions. In 2009, the first open source bitcoin client was released with Nakamoto mining the genesis block. In 2011, Satoshi Nakamoto distanced himself from the Bitcoin project claiming that it is in good hands [37] and until today his true identity has not been revealed.

In order for someone to use Bitcoin, first he has to connect to the Bitcoin network using one of the available clients. Despite the fact that our analysis is client-agnostic, we will use the Bitcoin core client as an example template because its source code has been heavily audited due to its open source nature.

By default, each client establishes a connection to eight other clients and start exchanging different types of information such as their current state, the block height, the transactions that have been relayed to them, cryptographic signatures, etc. Communication is carried out over a peer-to-peer network which topology is completely random even though each client maintains a local copy of potential addresses to perform initial connections to. In case it is the client's first ever attempted connection, thus no local list of peers exists, then the client uses one of the source code's hardcoded seed nodes. For the rest of his lifecycle, the client tries to maintain 8 outgoing connections, while at the same time accepts incoming connections from other nodes which are capped to 125 simultaneous connections. The information exchanged between two nodes is propagated through the entire network, an action necessary, as we will later see, for the validation of the transactions by the network.

3.2.1 Addresses and Transactions

As mentioned before, the core of the Bitcoin network is the blockchain that contains the balance of every Bitcoin user at any given time. However, instead of associating names with accounts, Bitcoin identifies its users by Bitcoin addresses.

A Bitcoin account is basically a public/private ECDSA (specifically secp256k1 [38]) keypair and a Bitcoin address is a 160-bit hash of the public portion of that keypair. The algorithms used for this one-way conversion are SHA256 and RIPEMD160 [39]:

$$A = RIPEMD160\ (SHA256\ (P_k))$$

Where A is the calculated Bitcoin address and P_k is the public key

When someone wishes to send Bitcoins from one account to another, he issues a transaction. This transaction is created by signing a hash of the transaction through which the Bitcoins were originated. Given that in Bitcoin there is one-to-one correspondence between public keys and addresses, a transaction between two addresses as and a_r has the following form [40]:

$$T(a_S \rightarrow a_R) = \{source,\ B,\ a_R,\ SIG_{sk_{as}}(source,\ B,\ a_R)\}$$

Where $SIG_{sk_{as}}$ is the signature which is using the signer's private key sk_{as} corresponding to the public key of address a_s, B the amount of Bitcoins and source is a reference to the transaction (the most recent one) that as acquired the Bitcoins from. Apparently, since everyone knows the public key of as, the validity of this signature can be independently verified by anyone.

In Bitcoin terms, spending is basically transferring value from a previous transaction, called transaction input, to a transaction output. A transaction input is where the Bitcoins are coming from and the transaction output declares a new owner for these Bitcoins by associating them with a key, called encumbrance. This output can then be used as an input for another transaction, hence a chain of ownership is created.

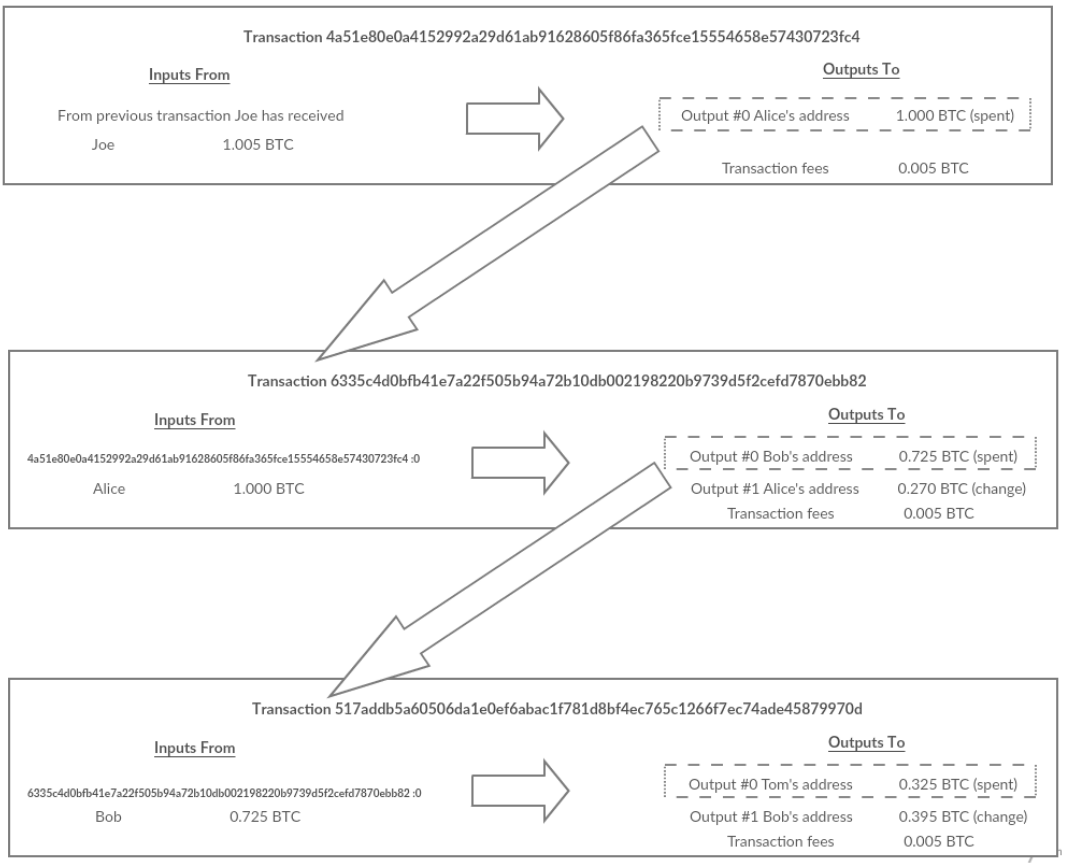

Figure 7 - A chain of transactions where the output of one transaction is the input for another

Transactions, in their most common form, have one input and one output. Since the value of an input cannot always match the exact value of an output, sometimes a new output is created which contains some change returning to the sending address.

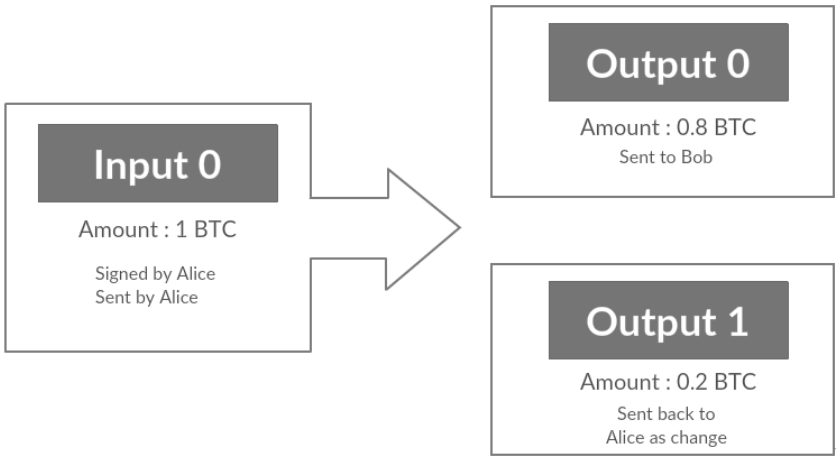

Figure 8: Common form of a bitcoin transaction with one input and one output

Another commonly used type of transaction is the one that combines multiple inputs into a single output. If we consider the previous example, the change received by multiple payments will result in multiple new smaller inputs, which will eventually have to be aggregated in one single output.

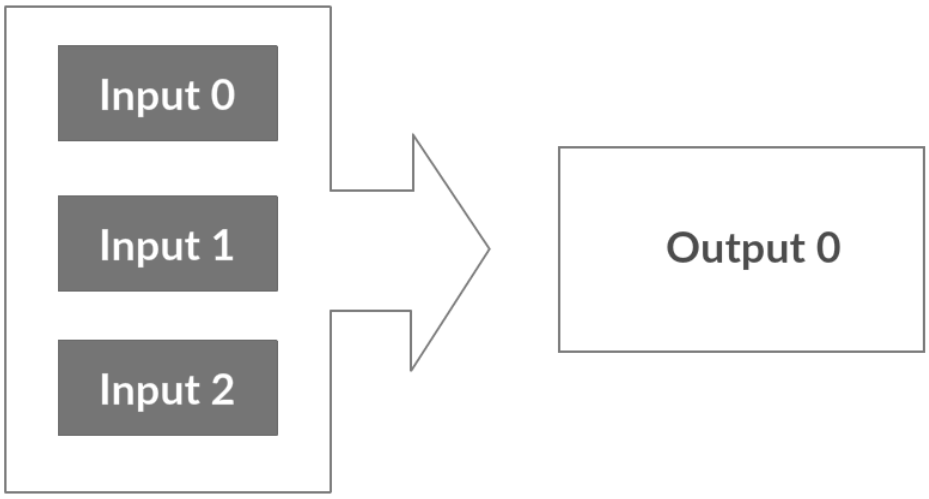

Figure 9: Combination of multiple inputs to one output

Finally, one last form of transactions often seen on Bitcoin's blockchain is a transaction that splits one input to N outputs which represent multiple Bitcoin adresses. It is commonly used by mining pools to distribute earnings among miners or business entities to process their payroll.

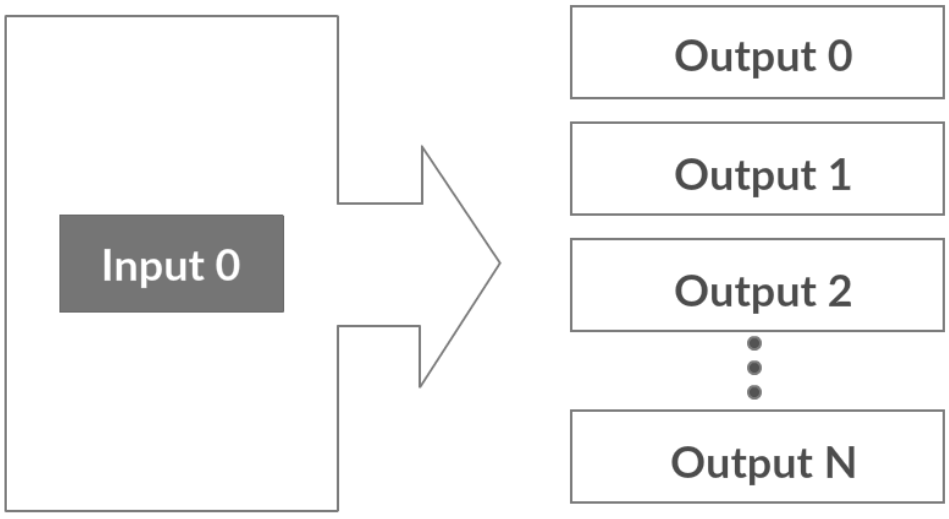

Figure 10: Common form of a transaction distributing funds across multiple outputs

3.2.2 Block Propagation in Bitcoin Network

When a transaction is transmitted to the network, it is then subjected to validity checks and it is not verified until it becomes part of the blockchain. New transactions constantly flow in the network and they get added to a memory pool of unconfirmed transactions handled by each node. Since the size of each block is finite, transactions have to deal with competition in order to be added in the new block and the selection criteria is based on who paid the highest fee.

As nodes build a new block, they add unconfirmed transactions from the memory pool to a new block and attempt to solve a computationally intensive problem to prove that the block is valid. This is the proof-of-work concept of Bitcoin and the process of solving it is called mining. Mining ensures that transactions are only confirmed if enough computational effort was spent on the blocks that contain them. More blocks mean more effort which subsequently means more trust [39].

To incentivize mining, each mining node includes a special transaction in its block containing a transaction that pays its own address a reward (currently 12.5 BTC per block) of newly created Bitcoins. If the node finds the solution before the other nodes in the network, then the block becomes valid, and it wins the reward since the block is added to the blockchain, thus the reward transaction becomes spendable. This reward transaction is the only exception to the rule that a transaction's outputs has to be smaller or equal to its inputs.

The block is then propagated throughout the network and contains a list of transactions that the node which created the block committed since the previous block [41]. To prevent denial-of-service attacks and spam, every node that receives this newly created block validates it before forwarding it further. If it determines that it is a valid block then it propagates it to its adjacent nodes, discards its previous mining efforts, applies the transactions from the current block and immediately starts working on building the next block.

At this point, the network has agreed on the validity of the transactions contained in the newly mined block and the transactions are confirmed and do not have to be reapplied. The transactions that were not included will have to be validated again and reapplied on top of the new block state.

3.3 How Zombiecoin Used Bitcoin's Network

The first work that proposed the use of a publicly available infrastructure and overlay a C&C communication channel on top of it, was Nappa et al. [42] who basically suggested a C&C channel overlaid on the Skype network. However, contrary to Bitcoin, Skype doesn't offer a decentralized environment, it is closed source and since its acquisition by Microsoft in 2011, Skype switched to a cloud based architecture [43].

S.T. Ali et al. [44] proposed a scheme which offered considerable advantages over existing C&C architectures and used the Bitcoin network as a leverage for communication. The comparative advantage of Zombiecoin, as the authors named the proposed bot, is multifactorial, however it can be summed up to the following.

First of all, it eliminates the need for botmasters to maintain complex and custom C&C networks. This allows them to be spared the cost, the hassle and more importantly the increased risk that is inherent in large scale botnet networks. Second, the authors of Zombiecoin, arguably claim that the Bitcoin network offers anonymity especially if combined with Tor and VPNs.

As a final advantage, the study reports the fact that the Bitcoin network is resilient to takedown attempts and any form of regulation, claims that will be further examined in this study.

3.4 What is Monero

Monero (XMR) is an open source cryptocurrency created in 2014 which primarily focuses on privacy, fungibility and scalability. One of the reasons that makes Monero unique is that unlike many other cryptocurrencies, it doesn't share a common codebase with Bitcoin but instead it is based on Cryptonote protocol [45], albeit with several modifications.

Monero gained publicity when, in August 2016, two of the largest darknet markets started accepting it [46], resulting in a 20x spike in its value. It also spiked interest of several Bitcoin maximalists and developers due to its strong focus on scalability, an issue plaguing Bitcoin.

Little improvements over the Bitcoin network protocol abound in Monero. It is still a peer-to-peer cryptocurrency implementing a random network topology based on constantly updating node lists. One difference between the two clients is that there is no hard cap in Monero's client regarding maximum number of outgoing connections. Instead, it implements a configurable bandwidth limit on ingress and egress traffic.

3.4.1 Addresses and Transactions

Monero's public addresses are quite different compared to Bitcoin. More specifically Monero makes use of two keypairs, the view keypair and the spend keypair. Contrary to Bitcoin, the view and spend keypairs are EdDSA [47] (specifically ed25519). The private spend key and the private view key are passed through the ed25519 scalar function to create their public counterparts.

To derive the public address from these keys, the following transformations happen:

1. One network byte is appended at the beginning of the pair of public keys resulting in a 65-byte string.
2. These 65 bytes are hashed with Keccak-256
3. The first four bytes from the hashed value are prepended to the 65-byte value of step 1, resulting in a 69-byte public address.
4. The 69-byte string is divided into 8-byte blocks which are separately converted to Base58, creating a 95-character string which is the Monero public address.

The transaction structure of Monero remains similar to Bitcoin in its basis, where the client selects several incoming transactions (inputs), signs them with his private key and sends them to the recipient. However, contrary to Bitcoin's model, where each user has a unique public and private key, Monero implements stealth addresses which basically require from the sender to create random addresses on behalf of the recipient, bound to be used only once. This way the recipient can publish his address, albeit have all his incoming transactions go to unique addresses in the blockchain without publicly revealing his transactions.

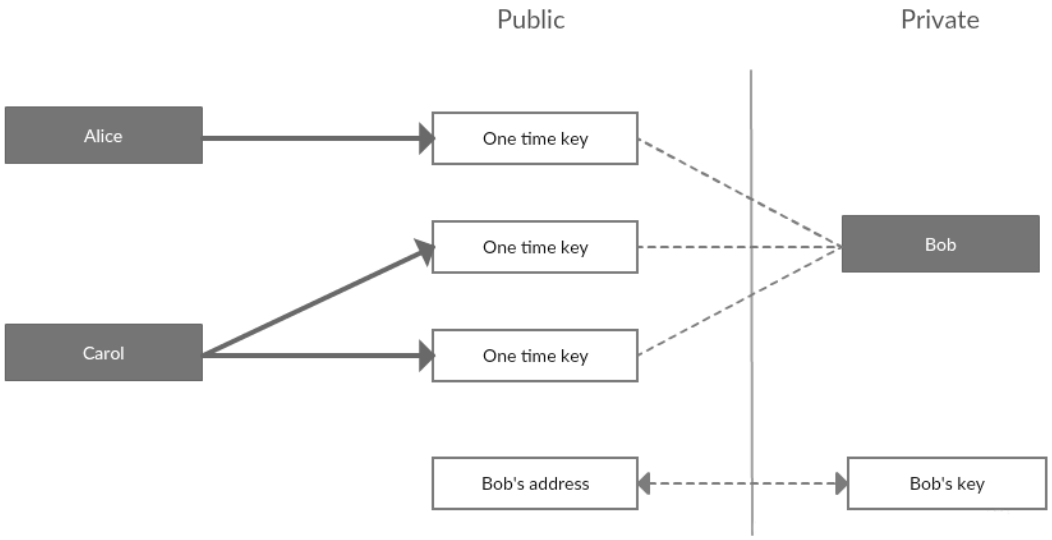

Figure 11: The keys/transaction model of Cryptonote

In a typical Monero transaction the sender performs a Diffie-Hellman exchange which allows him to get half of the recipient's data and a shared secret. Then, using the second half of the recipient's address and the shared secret, he calculates a one-time address. If we break down a Monero transaction, the detailed steps are as follows:

1. Alice wants to send Bob a payment. She decrypts Bob's address and gets Bob's public key (A, B).
2. She generates a random $r \in [1,\ l-1]$ and calculates a one-time public key P:

$$P = H_s\,(rA)G + B$$

3. She uses P as a destination address for the output and inserts R = rG in the transaction.
4. She sends the transaction.
5. Bob checks all transactions with his private key (a, b) and calculates:

$$P' = H_s\,(aR)G + B$$

6. If the transaction is intended for Bob, then $aR = arG = rA$ and $P' = P$
7. Bob recovers the corresponding one-time private key x:

$$x = H_s(aR) + b \leftrightarrow P = xG$$

H_s: a hash function $\{0,1\}^* \rightarrow F_q$

l: a prime of the base point, $l = 2252 + 27742317777372353535851937790883648493$

G: a base point in the elliptic curve $G = (x, -\frac{4}{5})$

a: a standard elliptic curve private key, $a \in [1, l-1]$

A: a standard elliptic curve public key, $A = aG$

(a, b): a private user key pair of two private elliptic curve keys

(a, B): a tracking key pair of private and public elliptic curve keys, $B = bG$ and $a \neq b$

On top of stealth addresses, Monero implements ring signatures to avoid tracing transaction outputs back to their respective senders and receivers. A ring signature is a group of cryptographic signatures with one at least real participant which makes possible to have a number of possible signers without having to reveal which of the group members actually created the signature [48]. In Monero terms, the ring size is called mixin and introduces a combinatorial explosion when applied to multiple transactions. For example, even with a mixin five, after 5 blockchain hops there will exist 3125 possible paths.

Monero's implementation of ring signatures uses four algorithms:

$$(GEN, SIG, VER, LNK)$$

GEN: Outputs an elliptic curve pair (P, x) and a public key I

SIG: Uses a message m as an input along with a set S' of public keys and a pair (P_s, x_s) and outputs a signature s and a set of public keys $S = S' \cup$ [49]

VER: Takes m, S and s as an input and outputs {true} or {false}

LNK: Takes set $I = \{I_i\}$ and signature s as an input and outputs {indep} or {linked}

The basic idea behind ring signatures is that a user creates a signature which can be verified by a group of public keys instead of just one public key allowing the signer to be indistinguishable from the other users participating in the ring.

Finally, as of January 10th 2017, Monero activated the use of ring confidential transactions, an idea originally proposed by Greg Maxwell who described confidential transactions as a way to send Bitcoins with the amounts hidden [50]. Maxwell's proposed model used a Pedersen commitment [51], a scheme that allows someone to choose a value while hiding it from the other participating parties. This commitment is binding in the sense that once committed to it, a party cannot change the chosen value.

Similar to Bitcoin, a transaction that is transmitted to the network, is subjected to validation checks and it is not verified until it becomes part of the blockchain. New transactions that enter the network are added to a memory pool of unconfirmed transactions handled by each Monero node individually. However, when a new block is mined and contrary to how Bitcoin behaves, the node will not send the whole block. Instead it will send a block header and a list of the transaction hashes included in that block. The receiving node will examine its pool of unconfirmed transactions and will just ask the transmitting peer for any transactions that it doesn't already have. This results in largely reduced bandwidth requirements since the blocks are propagated differentially.

Furthermore, instead of using a fixed block size, Monero utilizes a dynamic block size adjustment. More precisely, the minimum median block size is set to 60KB and the maximum block size at any given time is the maximum between 60KB and twice the median size of the last 100 blocks mined in the network. The purpose of this adjustment is that as the number of transactions increases, the block size will increase accordingly allowing some space scarcity to serve the increased demand. To prevent someone from spamming the network in order to artificially inflate the block size and overwhelm the network, Monero implements a quadratic penalty [52] to miners who create a block larger than the median size of the last 100 blocks and it is calculated as follows:

$$P = S_b \left[\left(\frac{B}{M}\right) - 1\right]^2 \text{ and } S = S_b - P$$

Where P is the penalty imposed on the miner, S the subside, S_b the base subsidy, B the current block size and M the median of the last 100 blocks.

Another feature that differentiates Monero from Bitcoin is the dynamic fees. Instead of using predetermined fees per KB of transaction, starting in version four of its blockchain, Monero calculates fees dynamically. The fees are based on the block size of a past time window and the current block reward. The formula that calculates it is:

$$F = \left(\frac{R}{R_0}\right)\left(\frac{M_0}{M}\right)F_0$$

Where R is the current block reward, R_0 the reference block reward (10 XMR), M the block size limit as calculated by the dynamic block formula, M_0 the minimum block size (60KB as previously mentioned) and finally F_0 is a constant which acts as a sanity check and is set to 0.002 XMR.

3.4.2 Disadvantages of Bitcoin as a C&C Communication Channel

Contrary to common belief, Bitcoin is not a truly anonymous coin. All the transactions in the network involve pseudonymous addresses and a user's transactions can easily be linked together. If somehow a transaction is linked to a user's true identity, then all of his transactions will be exposed since all transfers are permanently and globally visible in the blockchain. Even if the user uses a different address every time (creating a new address is free and trivial), recent papers have shown ways on how to link a user's different addresses and even his external identity [40] [53] [54].

Bitcoin in order to provide public verifiability, requires all of its transactions to be broadcasted in cleartext thus exposing a linkage between payees and payers to everyone. It is evident that this leads to very poor performance in the context of unlinkability and untraceability. Bitcoin's community is aware of this issue, leading to much discussion on how to provide stronger anonymity. Some of the suggested solutions are:

Mixing. It's the most common one and it's analogous to what mixes are in communications networks. In their most common form, mixers use a receiving address which receives coins from multiple users and then forwards them back in a random order back to a new address. However, mixers add a centralized point of failure in the network and in case they are compromised all the mixed coin's transaction graphs will be revealed, so they are deemed inadequate for a reliable.

Coinjoin. A slightly improved method, originally proposed by Greg Maxwell [55], which can be applied in a decentralized manner. However, with Coinjoin a user can run into liquidity problems and is susceptible to Sybil attacks. Also, as Kristov Atlas showed, if the protocol is implemented incorrectly, the level of anonymity is diminished [56].

Finally, despite Bitcoin's libertarian ideology, there have been many attempts recently that indicate that in the near future the pressure from governments will probably lead to a stricter regulatory framework and censorship. An example of such attempt is the recent press release by the U.S. Department of Treasury which calls the Bitcoin community to block two addresses that belong to two Iran-based individuals [57]. Monero on the other hand is highly unlikely to experience a similar problem since the addresses are not publicly available and its community considers illegal activities as an unavoidable side-effect of strong privacy and anonymity.

4 Monero as a Solution

Monero, as a platform, offers considerable advantages over Bitcoin with the most significant one being that it uses stealth addresses. As analyzed in 3.4.1 a user can publish his stealth address and sustain his anonymity since the destination address will be dynamically computed by the sender. This address will never reach the blockchain thus consisting him indistinguishable from the rest of the network and basically creating a huge anonymity set. To further enhance its anonymity and prevent passive surveillance, Monero uses I2P as a network layer [58]. I2P was chosen over Tor because I2P doesn't rely on directory services and also because of its symmetric design which allows more routers (basically every network participant is a router).

Furthermore, in Monero, nobody inspecting the blockchain is able to tell where the coins came from (even the recipient is unable to) and this is due to the use of the one-time ring signatures. As shown in [59] an attacker would require 87% of the unspent transaction outputs in order to identify 1% of the total transactions. Additionally, by combining ring signatures, stealth addresses and ring confidential transactions, Monero achieves complete unlinkability and untraceability, an invaluable feature for any botnet operation.

In this chapter we will propose a solution based on Monero's blockchain that uses freely available software and we will demonstrate how realistic and practical such a solution is.

4.1 Operation of our Proposed Bot

Prior to deploying the bots, the botmaster must create a Monero address which will be the address where all of his commands will be sent to. From this generated address, the botmaster can derive three keys, a secret and a public *viewkey* and a private spend key. Also, the botmaster has to create a 128 bit long AES key [49] that will used to encrypt the commands. The secret viewkey along with the AES key are embedded into the payload so that the bot will be able to decrypt any instructions sent by him.

The second step is what we described in 2.2 as the initial infection and the secondary injection. For the purpose of this work, we can combine them into one since the infection technique that will be used is not important. As soon as a host is infected, a unique identifier is created that will allow the botmaster to distinguish the bots. This identifier can also contain encoded information regarding the victim's operating system, hardware information or even geolocation data, which are extremely valuable information for any botmaster.

In the third step, the bot is required to connect to the Monero network and begin propagating transactions. For this step, we can follow two different approaches. The bot can either connect to a remote node that hosts the whole Monero blockchain or the bot can begin downloading the blockchain locally, hence becoming a full node for the Monero network. The advantage of the first approach is that the bot is synchronized with the network almost immediately and becomes ready to receive instructions in a very short amount of time. Additionally, since Monero's blockchain size as of today is around 68GB, the synchronization with the network is a resource intensive procedure both storage and bandwidth wise. However, this last issue will be probably resolved in the future by what is known as blockchain pruning, which will allow every peer on the network to start synchronizing from a specific block height. Therefore, if a bot is deployed in 2018, there is no need to download the whole blockchain from 2014, but instead download only the last blocks. A more fine-grained approach would also allow this block height to be calculated dynamically, by reading the timestamp of the host at the time of the infection, and automatically determine the most recent block. This way, the storage and bandwidth requirements will be kept at the bare minimum. The main disadvantage of the remote node approach is that we introduce a centralized element in an architecture which in all other aspects is completely decentralized. In any case, it is possible for both the remote and full node methods to be combined in such way that they become a failback for each other. If the synchronization with the network stops for some reason, the bot can switch to a remote node and vice versa.

In the final step, the botmaster issues commands to the botnet using Monero's blockchain as a C&C communication channel. To accomplish that, he creates a transaction and sends it to the address specified in the first step. This transaction contains a *payment id* which is a 32-byte hexadecimal string, encrypted with the private spend key, which can only be decrypted by the bots using the secret view key.

4.2 Testing Environment

To evaluate our solution, we created a network with 4 machines. One played the role of botmaster's main computer, the other one was running a daemon in RPC mode ready to receive commands from the botmaster and propagate them in the network and the last two computers are the bots which are connected to the Monero network. For demonstration purposes, we decided to run one computer as a full node and the other node was connected to a remote node.

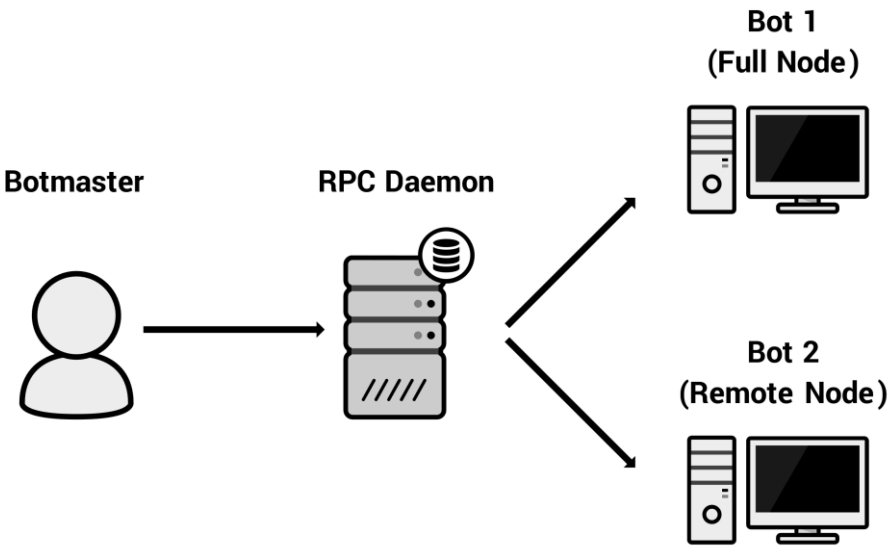

Figure 12

Our testing environment consists of four virtual machines created in VirtualBox with their network adapters set in NAT mode. We used the latest Monero client version 0.13.0.4 and for the testing script, Python 3.7 was used. Also, instead of using livenet, we used testnet. In principal they are exactly the same, the latter however allows us to perform our tests without having to purchase Monero.

First of all, we created an address which will act as the rendezvous point [15] for all bots. This address will be constantly monitored by the bots and will receive transactions that will contain commands issued by the botmaster. The keys and the address created for our demonstration, can be found at Table 2.

We assume that the botmaster already has a Monero address with enough funds ready to be spent, even though a small amount is required as we will later see.

Finally, we use the openssl library to generate our AES key which will be later used for the encryption of our commands. The command used for the creation of the key is the following:

```
enc -aes-128-cbc -k dissertation -P -md sha1
```

And the result is the following:

```
salt=4E9B6D2C10B7D59A
key=D8EEB29F92397A6C8F2EAFA6E560E7A2
iv =29340AD92F86721EEDF2103238536BFF
```

Table 2

Botnet Keys

Address	9tog6A6dAeF34PEygcJYMs8yr5TskiwS8BidjtpHqyUrF4WgX84SxfwbzmbzxuhesvWxSdMfT18tc6d2fmfij2QKSTeU4Fy
Secret Viewkey	54697c65c7faee301b1e3b77d8c0647bb65cab850124b1443a52dff681450b0f
Public Viewkey	10015444d8e006d13a3eb1e8947bddb31836e33d686b0d219b5c087560a94ce1
AES Key	D8EEB29F92397A6C8F2EAFA6E560E7A2

4.3 Sending Commands

We will be using the *payment id (Pid)* feature of Monero in order to enter C&C commands in a Monero transaction. Payment id is an arbitrary and optional message which can be attached by the sender to any transaction. According to the specifications of Monero's protocol, it can be either a 32-byte unencrypted hexadecimal string or an 8-byte string, embedded into what is called an intergrated_address. The difference between the two types of payment ids is that the unencrypted ones are visible in a blockchain explorer, whereas the encrypted ones remain hidden. However, even when a payment id is visible, blockchain analysis cannot reveal any other information about the transaction or distinguish a Pid containing C&C commands from any other legit Pid.

We will be using the 32-byte variant of the pid which offers us more bandwidth to include commands and its structure is depicted in Figure 13

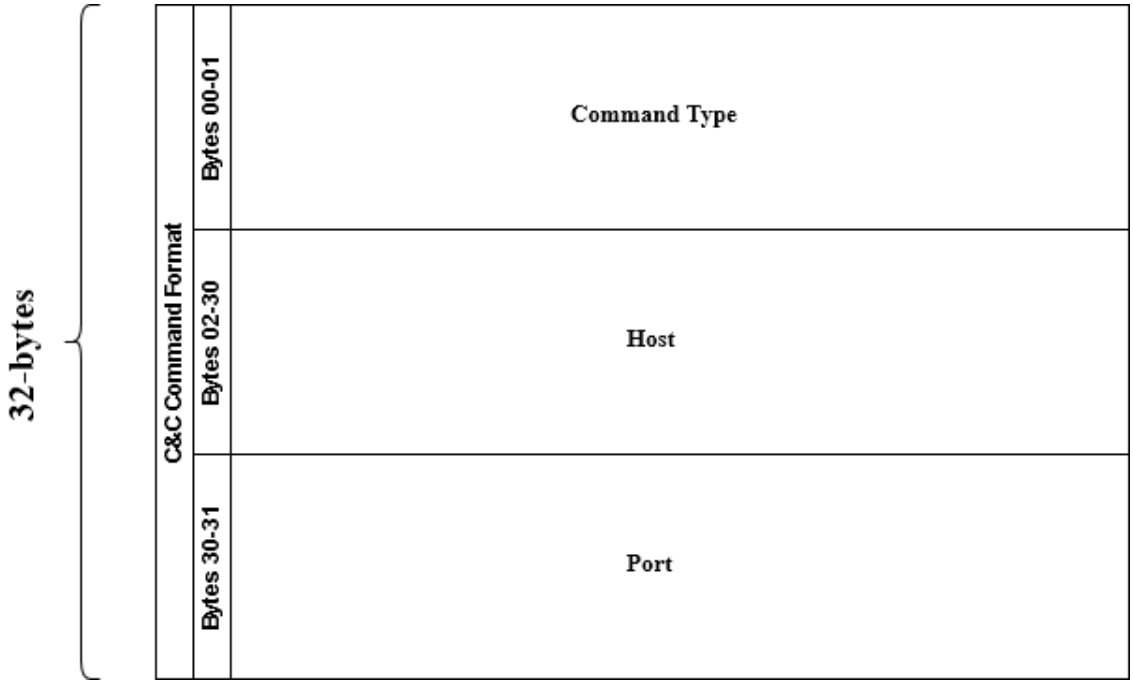

Figure 13

The first two bytes are used as an identifier of the type of the command that the bot has to execute, the next 28 bytes represent the host that the command will apply to and finally the last two bytes correspond to the port that will used. Some of the most common commands that bots use are Ping, Update, Download, DDOS, Screenshot, Scan, and Upload. Since we have two bytes available, we can assign each individual command to a hex value. Some indicative commands are shown in Table 3, however we can assign up to 65535 different commands given the two-byte capacity.

Table 3

Command Hex Values

Start Ping	00 00
Stop Ping	00 02
HTTP Update	00 04
HTTP Download	00 06
FTP Download	00 08
Start DDOS	00 0A
Stop DDOS	00 0C
Screenshot	00 0E

Start Scan	00 10
Stop Scan	00 12
Upload	00 14

The next 28 bytes are used to assign the host that the command applies to. The host field can either be a target, if the command is an attack, or it can be a server that hosts an updated payload. Finally, the port that will be used for the connection is determined by the last four bytes of the pid.

4.4 Proof of Concept

As we mentioned in 4.1, the first step for the botmaster is to connect to an RPC server which will relay the transactions to the network. In order to run the wallet in RPC mode, we issue the following command on the server:

```
monero-wallet-rpc.exe --testnet --rpc-bind-port 18082 --wallet-file
botnet --password "" --daemon-address monero-testnet.exan.tech:28081
```

--testnet: Indicates that the wallet will synchronize with the testnet blockchain

--rpc-bind-port: The port that the RPC server listens

--wallet-file: The wallet containing the funds that will be used for the transaction

--daemon-address: The address of the remote node that the server will relay the transactions in order to propagate to the rest of the network.

To ensure that communication with the server is possible, we can execute the following Python script:

```
import requests
import json


class GetBalance():

    def __init__(self):
        # Initial Connection
        self.url = "http://localhost:18082/json_rpc"
```

```
        # standard json header
        self.headers = {'content-type': 'application/json'}

    def get_balance(self):
        """return the wallet's balance"""

        rpc_input = {
            "method": "getbalance"
        }

        response = self.__do_rpc(rpc_input)

        return response.json()

    def execute(self):

        example_functions = [self.get_balance]

        for fun_obj in example_functions:
            print(fun_obj.__name__ + "():\n" + fun_obj.__doc__)
            json_result = fun_obj()
            print(json_result, "\n")

    def __do_rpc(self, rpc_input):

        rpc_input.update({"jsonrpc": "2.0", "id": "0"})

        # execute the rpc requrest
        response = requests.post(
            self.url,
            data=json.dumps(rpc_input),
            headers=self.headers)

        return response


if __name__ == "__main__":
    sw = GetBalance()
    sw.execute()
```

The result is a JSON object like the following:

[{'address': '9wufLD1qpK5YoUchJ4FbFWFr4D22ZCtqD11ThpGwxN4j6eG-hxvvdrJ2YRxdpNTURYraMoz8FkmhBnM9WfwNRe4zd9YiUtzu', 'balance': 87508031318408}]

Which means that the server we just connected to, has approximately 87 coins in its wallet's balance, but more importantly accepts connections.

The next step is for the bots to start listening for new transactions. In order to do so, we are going to use the secret view key that we generated in 4.2. The command is the following:

```
botnet_listener.exe --testnet --generate-from-view-key botnet --dae-
mon-address monero-testnet.exan.tech:28081
```

--testnet: Indicates that the wallet will synchronize with the testnet blockchain

--generate-from-view-key: Creates a wallet that has permission only to view incoming transactions

--daemon-address: The address of the remote node that the bot will listen for new transactions.

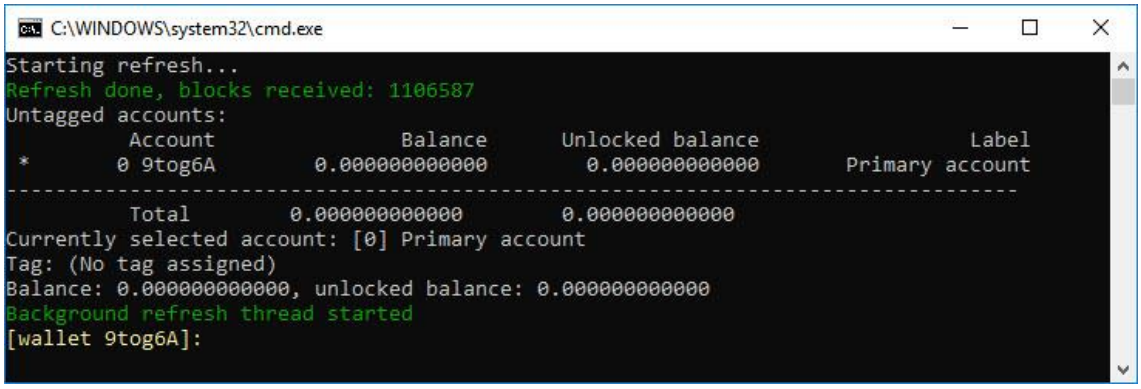

Figure 14 – The synchronized wallet, waiting for incoming transactions

To simulate an actual command, let's assume that we want to perform a DDoS attack against the web server that hosts the University's website. According to Table 3, the corresponding hex code to start a DDoS attack is 00 0A. Furthermore, by converting ihu.edu.gr to a hexadecimal value we get 69 68 75 2e 65 64 75 2e 67 72 which is 10 bytes long. To ensure compliance with Monero's protocol which requires a 32-byte pid, we pad this value with leading zeros which becomes 00 00 00 00 00 00 00 00 00 00 00 00 00 00

00 00 00 00 69 68 75 2e 65 64 75 2e 67 72. Finally, ihu.edu.gr uses https, which means that the attack has to be against port 443 or 01 BB in hex.

We end up *with 00 0A 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 69 68 75 2e 65 64 75 2e 67 7201 BB* as our pid number. However, as we mentioned before, the pid will be visible in the blockchain, and since converting a hexadecimal to text is something trivial, we need to encrypt it before transmitting it.

For this purpose, we are going to use the AES key that we created in 4.2. At this point it is worth noticing that AES-128-CBC supports 16-byte blocks, hence we are guaranteed that we will get a 32-byte output as long as we provide a 32-byte input. We encrypt the pid in OpenSSL and we end up with a final value of *9c d5 00 fe 23 40 05 35 9a 30 bb 70 ed f5 27 32 4a db d1 4f f9 8e e7 f2 40 42 b0 3d 02 61 97 1e,* which is our final pid.

The botmaster can now issue his C&C command with following Python script:

```
import requests
import json
import os


def main():

    # wallet is running in RPC mode
    url = "http://<server_address>:18082/json_rpc"

    # standard json header
    headers = {'content-type': 'application/json'}

    destination_address =
"9tog6A6dAeF34PEygcJYMs8yr5TskiwS8BidjtpHqyUrF4WgX84SxfwbzmbzxuhesvWxS
dMfT18tc6d2fmfij2QKSTeU4Fy"

    # send specified xmr amount to the given destination_address
    recipents = [{"address": destination_address,
                  "amount": 1000000}]

    payment_id =
"9cd500fe234005359a30bb70edf527324adbd14ff98ee7f24042b03d0261971e"

    # simplewallet' procedure/method to call
    rpc_input = {
```

```
        "method": "transfer",
        "params": {"destinations": recipents,
                   "payment_id" : payment_id}
    }

    # add standard rpc values
    rpc_input.update({"jsonrpc": "2.0", "id": "0"})

    # execute the rpc request
    response = requests.post(
         url,
         data=json.dumps(rpc_input),
         headers=headers)

    # print the payment_id
    print("#payment_id: ", payment_id)

    # pretty print json output
    print(json.dumps(response.json(), indent=4))


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

The transaction enters the memory pool of the blockchain and we can almost immediately see it in the bot's wallet by issuing the *show_transfers in* command. We immediately also see it in a public explorer but since pid is encrypted, it does not raise any suspicions.

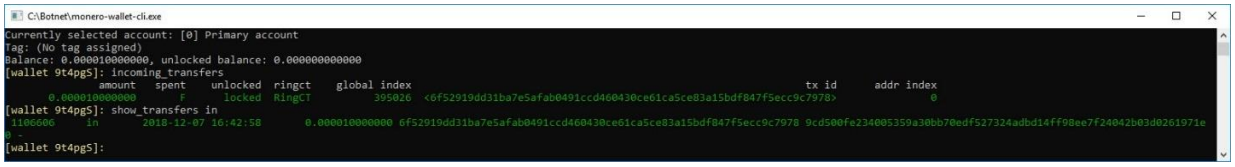

Figure 15 – The result of the *show_transfers in* on the bot's side

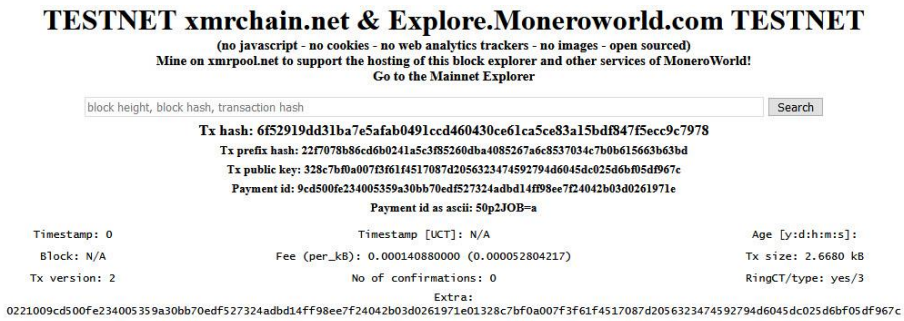

Figure 16 – The transaction as shown in a blockchain explorer

From this point, the bot can apply the reverse technique and retrieve the C&C command. More specifically, it decodes the pid of the received transaction using the AES key and then breaks down the resulting hex according to the structure of Figure 13.

4.5 Cost of Operation

In our proposal, the total expenses can be pinpointed down to two factors. The first one is the amount sent by the botmaster to the address of the bots and the second one is the fee that the network charges every time a transaction is made. Since the amount that is sent during the transaction goes back to a wallet controlled by the botmaster, we will only examine the cost of the fees.

Even though in Monero the fee is dynamically calculated, we can have a quite good estimate by studying a blockchain explorer. As shown in Figure 16, the transaction size was 2.66 kilobytes and the total fee paid was 0.00014 XMR, which with today's exchange rates amounts to 0.005€. However, we have to keep in mind that our test was performed on testnet, where blocks are usually empty thus significantly increasing the fee per kB. Examining a livenet blockchain explorer, reveals that the cost of a similarly sized transaction would have been undoubtedly lower. More specifically, at the time of the transaction of our test, a 2.65kB transaction on livenet would have cost 0.00006 XMR, or 0.00222€.

In the case of Bitcoin, a typical transaction with one input and two outputs is around 250 bytes. However, although the average transaction size in Bitcoin is lower, that doesn't apply to the transaction fees as well. The cost for a similar transaction at the time of writing, would have been around 0.001 BTC or 2.8€.

If we take into consideration the fact that our proposal does not require any other expenditure in infrastructure, specialized software or maintenance costs, it becomes apparent that, at least from a logistics point of view, Monero's blockchain as a C&C mechanism is a very tempting solution.

5 Conclusions

In this dissertation, we tried to establish a framework for evaluating the challenges of maintaining a botnet from a botmaster's perspective. We examined the evolution of botnets in a historical context and we presented the most common architectures used by them. Then, we examined the most popular techniques used for their mitigation, something that subsequently rises a challenge for every botmaster, hence leading malware authors in search of new opportunities and developments.

We presented a solution based on Monero's blockchain and demonstrated through a proof of concept that such a solution is not only feasible but, also considering all the advantages that Monero offers in terms of anonymity and unlinkability, very hard to countermeasure. It offers a completely decentralized platform and contrary to Zombiecoin's implementation, it is almost impossible to censor or trace through blockchain analysis. Furthermore, it offers extreme resilience to traditional takedown attempts and it is very cost effective.

We strongly believe that botnets based on blockchains are going to become more popular in the near future and they are going to be a handful for government and organizations, especially if implemented correctly.

Bibliography

1. Bacher, P., et al., *Know your enemy: Tracking botnets.* The Honeynet Project & Research Alliance, 2005.
2. Cooke, E., F. Jahanian, and D. McPherson, *The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets.* SRUTI, 2005. **5**: p. 6-6.
3. Ramachandran, A. and N. Feamster. *Understanding the network-level behavior of spammers*. in *ACM SIGCOMM Computer Communication Review*. 2006. ACM.
4. Eason, G., B. Noble, and I. Sneddon, *On Certain integrals of EggDrop: Open source IRC bot, 1993*.
5. Silva, S.S., et al., *Botnets: A survey.* Computer Networks, 2013. **57**(2): p. 378-403.
6. Kharouni, L., *SDBOT IRC botnet continues to make waves.* A Trend Micro White Paper, 2009: p. 1-20.
7. Holz, T., *A short visit to the bot zoo [malicious bots software]*. IEEE Security & Privacy, 2005. **3**(3): p. 76-79.
8. Grizzard, J.B., et al., *Peer-to-Peer Botnets: Overview and Case Study.* HotBots, 2007. **7**: p. 1-1.
9. Macesanu, G., et al. *Development of GTBoT, a high performance and modular indoor robot*. in *Automation Quality and Testing Robotics (AQTR), 2010 IEEE International Conference on*. 2010. IEEE.
10. Yen, T.-F. and M.K. Reiter. *Traffic aggregation for malware detection*. in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. 2008. Springer.
11. Bailey, M., et al. *Automated classification and analysis of internet malware*. in *International Workshop on Recent Advances in Intrusion Detection*. 2007. Springer.
12. Royal, P., *Analysis of the kraken botnet.* Damballa, Apr, 2008. **9**.
13. Chiang, K. and L. Lloyd, *A Case Study of the Rustock Rootkit and Spam Bot.* HotBots, 2007. **7**: p. 10-10.

14. Holz, T., et al., *Measurements and Mitigation of Peer-to-Peer-based Botnets: A Case Study on Storm Worm.* LEET, 2008. **8**(1): p. 1-9.
15. Porras, P.A., H. Saïdi, and V. Yegneswaran, *A Foray into Conficker's Logic and Rendezvous Points.* LEET, 2009. **9**: p. 7.
16. Group, C.W. *Conficker working group: Lessons learned*. 2011; Available from: http://www.confickerworkinggroup.org/wiki/uploads/Conficker_Working_Group_Lessons_Learned_17_June_2010_final.pdf.
17. Matrosov, A. and E. Rodionov. *Festi botnet analysis and investigation*. in *Proc. 15th AVAR Conf*. 2011.
18. Matrosov, A., E. Rodionov, and D. Harley. *TDSS part 1: The x64 Dollar Question*. 2011; Available from: https://resources.infosecinstitute.com/tdss4-part-1/.
19. Andriesse, D. and H. Bos, *An analysis of the zeus peer-to-peer protocol.* VU University Amsterdam, Tech. Rep. IR-CS-74, 2013.
20. Abu Rajab, M., et al. *A multifaceted approach to understanding the botnet phenomenon*. in *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*. 2006. ACM.
21. Barsamian, A.V., *Network characterization for botnet detection using statistical-behavioral methods*. 2009, Dartmouth College.
22. Schiller, C. and J.R. Binkley, *Botnets: The killer web applications*. 2011: Elsevier.
23. Liu, L., et al. *Bottracer: Execution-based bot-like malware detection*. in *International Conference on Information Security*. 2008. Springer.
24. Choi, H., et al. *Botnet detection by monitoring group activities in DNS traffic*. in *Computer and Information Technology, 2007. CIT 2007. 7th IEEE International Conference on*. 2007. IEEE.
25. Gu, G., et al., *Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection.* 2008.
26. Ollmann, G., *Botnet communication topologies.* Retrieved September, 2009. **30**: p. 2009.
27. Jelasity, M. and V. Bilicki, *Towards Automated Detection of Peer-to-Peer Botnets: On the Limits of Local Approaches.* LEET, 2009. **9**: p. 3.
28. Lua, E.K., et al., *A survey and comparison of peer-to-peer overlay network schemes.* IEEE Communications Surveys & Tutorials, 2005. **7**(2): p. 72-93.

29. Memon, G., J. Li, and R. Rejaie, *Tsunami: A parasitic, indestructible botnet on kad.* Peer-to-Peer Networking and Applications, 2014. **7**(4): p. 444-455.
30. Chang, S., et al. *A framework for p2p botnets*. in *Communications and Mobile Computing, 2009. CMC'09. WRI International Conference on*. 2009. IEEE.
31. Yadav, S. and A.N. Reddy. *Winning with DNS failures: Strategies for faster botnet detection*. in *International Conference on Security and Privacy in Communication Systems*. 2011. Springer.
32. Williams, J. *Operation b107–Rustock Botnet Takedown*. Microsoft Malware Protection Center, Mar 2011 [cited 17; Available from: https://blogs.technet.microsoft.com/microsoft_blog/2011/03/17/taking-down-botnets-microsoft-and-the-rustock-botnet/.
33. Sood, A.K. and R.J. Enbody, *Crimeware-as-a-service—a survey of commoditized crimeware in the underground market.* International Journal of Critical Infrastructure Protection, 2013. **6**(1): p. 28-38.
34. Chaum, D., *Blind Signatures for Untraceable Payments*, in *Advances in Cryptology*. 1983, Springer US. p. 199-203.
35. Chaum, D., A. Fiat, and M. Naor, *Untraceable electronic cash.* Advances in Cryptology — CRYPTO 88, 1990: p. 319-327.
36. Nakamoto, S., *Bitcoin: A Peer-to-Peer Electronic Cash System*. 2009: https://bitcoin.org. p. 9.
37. Popper, N., *Digital Gold: The Untold Story of Bitcoin*. 2015: Allen Lane.
38. Malvik, A.G. and B. Witzoee, *Elliptic Curve Digital Signature Algorithm and its Applications in Bitcoin.* 2015.
39. Antonopoulos, A.M., *Mastering Bitcoin*. 2014: O'Reilly Media.
40. Androulaki, E., G.O. Karame, and M. Roeschlin, *Evaluating User Privacy in Bitcoin,* in *17th International Conference on Financial Cryptography and Data Security* A.-R. Sadeghi, Editor. 2013, Springer: Japan p. 34-51.
41. Decker, C. and R. Wattenhofer, *Information propagation in the bitcoin network,* in *IEEE P2P 2013 Proceedings*. 2013, IEEE. p. 1-10.
42. Nappa, A., et al. *Take a deep breath: a stealthy, resilient and cost-effective botnet using skype*. in *International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment*. 2010. Springer.

43. Whittaker, Z. *Skype ditched peer-to-peer supernodes for scalability, not surveillance*. 2013; Available from: https://www.zdnet.com/article/skype-ditched-peer-to-peer-supernodes-for-scalability-not-surveillance/.
44. Ali, S.T., et al., *ZombieCoin 2.0: managing next-generation botnets using Bitcoin.* International Journal of Information Security, 2018. **17**(4): p. 411-422.
45. Saberhagen, N.v., *CryptoNote v 2.0*. 2013.
46. *Alphabay Market launched first phase of Monero implementation*. 2016 [cited 2017 08 January]; Available from: https://redd.it/4z0wm3.
47. Bernstein, D., D. Niels, and L. Tanja, *High-speed high-security signatures.* Journal of Cryptographic Engineering, 2012. **2**(2): p. 77-89.
48. Ronald, R., S. Adi, and T. Yael. *How to leak a secret*. in *International Conference on the Theory and Application of Cryptology and Information Security*. 2001. Springer.
49. Standard, N.-F., *Announcing the advanced encryption standard (AES).* Federal Information Processing Standards Publication, 2001. **197**: p. 1-51.
50. Maxwell, G. *Confidential Transactions*. 2015 [cited 2017 15 January]; Available from: https://people.xiph.org/~greg/confidential_values.txt.
51. Pedersen, T. and B. Pedersen, *Explaining gradually increasing resource commitment to a foreign market.* International business review, 1998. **7**(5): p. 483-501.
52. Spagni, R. *Monero project*. 2016 [cited 2017 15 January]; Available from: https://github.com/monero-project/monero/blob/master/src/cryptonote_core/cryptonote_basic_impl.cpp.
53. Ron, D. and A. Shamir. *Quantitative analysis of the full bitcoin transaction graph*. in *International Conference on Financial Cryptography and Data Security*. 2013. Springer.
54. Meiklejohn, S., et al. *A fistful of bitcoins: characterizing payments among men with no names*. in *Proceedings of the 2013 conference on Internet measurement conference*. 2013. ACM.
55. Maxwell, G. *CoinJoin: Bitcoin privacy for the real world*. 2013 [cited 2017 13 January]; Available from: https://bitcointalk.org/index.php?topic=279249.
56. Atlas, K. *Weak Privacy Guarantees for SharedCoin Mixing Service*. 2015 [cited 2017 13 January]; Available from: http://www.coinjoinsudoku.com/advisory/.

57. Treasury, U.S.D.o. *Treasury Designates Iran-Based Financial Facilitators of Malicious Cyber Activity and for the First Time Identifies Associated Digital Currency Addresses* Available from: https://home.treasury.gov/news/press-releases/sm556.
58. Zantout, B. and R. Haraty. *I2P data communication system*. in *Proceedings of ICN*. 2011.
59. Noether, S., S. Noether, and A. Mackenzie, *A Note on Chain Reactions in Traceability in CryptoNote 2.0*. 2014.