



YAŞAR UNIVERSITY  
GRADUATE SCHOOL OF NATURAL AND APPLIED SCIENCES

MASTER THESIS

**BOTNET IMPLEMENTATION OVER TOR KIND  
NETWORKS**

HÜSEYİN YAĞCI

THESIS ADVISOR: ASSOC. PROF. AHMET KOLTUKSUZ, PH. D.

DEPARTMENT OF COMPUTER ENGINEERING

PRESENTATION DATE: 15.05.2017

BORNOVA / İZMİR  
MAY 2017

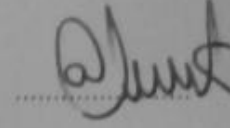


We certify that we have read this thesis and that in our opinion it is fully adequate, in scope and in quality, as a thesis for the degree of Master of Science.

**Jury Members:**

Assoc. Prof Dr. Ahmet KOLTUKSUZ  
(Supervisor)  
University: YAŞAR University

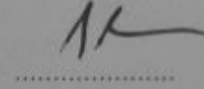
**Signature:**

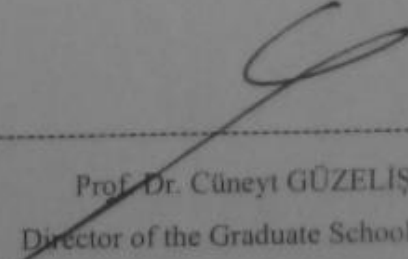


Asst. Prof. Dr. Mutlu BEYAZIT  
University: YAŞAR University



Asst. Prof. Dr. Tuğkan TUĞLULAR  
University: İZMİR INSTITUTE OF  
TECHNOLOGY



  
Prof. Dr. Cüneyt GÜZELİŞ  
Director of the Graduate School



## ABSTRACT

### BOTNET IMPLEMENTATION OVER TOR KIND NETWORKS

Yağcı, Hüseyin

MSc, Computer Engineering

Advisor: Assoc. Prof. Ahmet Koltuksuz, Ph. D.

May 2017

In this study, a novel implementation of a hidden botnet mechanism over networks similar to The Onion Routing (TOR) one is presented. The focus is on creating little sub networks with TOR-like structures, and hiding the existence of the botnets in the blind range of the internet. A proof of this concept's implementation and the details of the work were developed in a Linux environment. In doing so, this thesis contributes to the state-of-the-art networks by providing additional insights and a deeper understanding.

**Key Words:** Cyber warfare, Cyber security, Botnet, TOR, Anonymity, Network security



## ÖZ

### TOR YAPISI AĞLAR ÜZERİNDE BOTNET UYGULAMASI

Yağcı, Hüseyin

Yüksek Lisans, Bilgisayar Mühendisliği

Danışman: Doç. Dr. Ahmet Koltuksuz

Mayıs 2017

Çalışmada, anonimlik sağlayan ağlar kullanılarak kurgulanan yeni bir sistem ortaya konmaktadır. The Onion Routing (Tor) ağları üzerinde çalışabilen gizli bir botnet mekanizmasının özgün bir uygulaması gerçekleştirilmiştir. Araştırma, Tor benzeri yapılarla alt ağlar oluşturmak ve botnet'lerin varlığını internetin kör noktalarına gizlemek üzerine odaklanmaktadır. Bunu yaparken, tez, ek bilgiler ve daha derin bir anlayış sağlayarak çalışma alanına katkıda bulunmaktadır.

**Anahtar Kelimeler:** Siber savaş, Siber güvenlik, Botnet, TOR, Anonimite, Ağ güvenliği





## **ACKNOWLEDGEMENTS**

First of all, I would like to thank my supervisor Ahmet KOLTUKSUZ for his guidance and patience during this study.

I would like to thank aęatay YÜCEL for his constructive, collimating comments and support.

I would like to express my enduring love to my parents, who are always supportive, loving and caring to me in every possible way in my life.

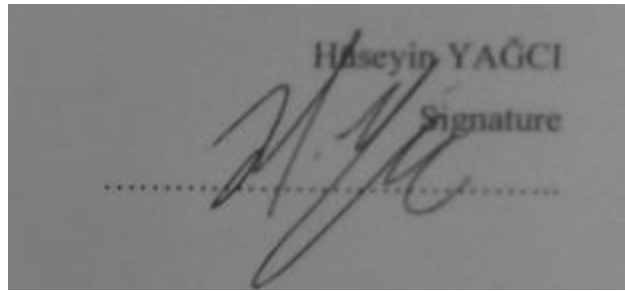
Hüseyin YAĞCI

İzmir, 2017



## TEXT OF OATH

I declare and honestly confirm that my study, titled “BOTNET IMPLEMENTATION OVER TOR KIND NETWORKS” and presented as a Master’s Thesis, has been written without applying to any assistance inconsistent with scientific ethics and traditions. I declare, to the best of my knowledge and belief, that all content and ideas drawn directly or indirectly from external sources are indicated in the text and listed in the list of references.



Huseyin YAGCI  
Signature

May 29, 2017



## TABLE OF CONTENTS

<b>ABSTRACT</b> .....	v
<b>ÖZ</b> .....	vii
<b>ACKNOWLEDGEMENTS</b> .....	ix
<b>TEXT OF OATH</b> .....	xi
<b>TABLE OF CONTENTS</b> .....	xiii
<b>LIST OF FIGURES</b> .....	xv
<b>SYMBOLS AND ABBREVIATIONS</b> .....	xvi
<b>CHAPTER ONE INTRODUCTION</b> .....	17
1.1. Scope .....	18
1.2. Motivation and Aim .....	18
1.3. This Work’s Novelty .....	18
1.4. Literature Review .....	18
1.4.1. Researches Based on Botnets.....	19
1.4.2. Researches Based on Tor network .....	24
<b>CHAPTER TWO BACKGROUND</b> .....	27
2.1. Botnet .....	27
2.1.3. Definition and Anatomy of a Botnet.....	27
2.1.4. Detection & Defense .....	31
2.2. Tor Network .....	32
2.2.1. Anatomy & Definitions of Tor Network.....	32
2.2.2. Vulnerabilities of Tor Network Anonymity .....	35
<b>CHAPTER THREE IMPLEMENTATION</b> .....	39
3.1. General Overview .....	39
3.2. Architecture and Structure .....	42
3.3. Vulnerability Assessment.....	43
<b>CHAPTER SIX DISCUSSION AND CONCLUSION</b> .....	45
<b>REFERENCES</b> .....	46
<b>APPENDIX 1 – System Setup Instructions</b> .....	49

<b>APPENDIX 2 – SAMPLE CODES .....</b>	<b>51</b>
<b>APPENDIX 3 – Tor configuration samples .....</b>	<b>54</b>
<b>APPENDIX 4 – Open Source Programs.....</b>	<b>67</b>

## LIST OF FIGURES

<b>Figure 1.</b> General Network Schema .....	40
<b>Figure 2.</b> Operation Communication Channel Schema.....	41
<b>Figure3.</b> General System Schema .....	42
<b>Figure 4.</b> Detailed Implemented Botnet-over-Tor Schema.....	43

## **SYMBOLS AND ABBREVIATIONS**

### ABBREVIATIONS:

TOR The Onion Routing

IRC Internet Relay Chat

LXC Linux container

PC personal computer

C&C Command and Control

DGA Domain Generation algorithm

APT Advance persistent Treat

APK Android Package Kit

API Application Programming Interface

IRC Internet Relay Chat

DoS Denial of Service

DDoS Distributed Denial of Service

SOCKS Secure Socket

TLS Transport Layer Security

RP Rendezvous Points

HS Hidden Service

HSD Hidden Service Directory



## **CHAPTER ONE**

### **INTRODUCTION**

This thesis strives to investigate the novel and extraordinary uses of The Onion routing (TOR) structure. This study distinguishes itself from most of the work on TOR which aims to increase the efficiency of known models; the emphasis here is to gain understanding on possible abnormal usages in anonymous TOR environments. As a direct consequence of desiderata, much investigative work was needed before finally developing the ideas presented here. This thesis requires a considerable amount of self-teaching in related areas, such as regarding the TOR network structure, anonymity approach, botnet approaches, threat intelligences, Darknet/Deep web approaches, Internet Relay Chat (IRC)-type communication protocols, information security, and computational cryptography.

The specific area of research here is an anonymized communication medium. The emphasis is not on the extension of the existing models, but in obtaining further insights into, and a deeper understanding of, abnormal usages of a TOR structure. In order to make this study demonstrable, a simple botnet design and its implementation have been worked on to fit the emphasized anonymity desire. The design of the botnet in this thesis is based on, and works with, centralized IRC protocols.

An applied communication medium works over a specialized TOR. The structure of the network was configured to be separate from the original TOR directories. In this manner, separated and specialized TOR structures can be integrated into the massive TOR network, but the original TOR network cannot discover these kinds of special TOR structures. Due to the different keys and differently configured directory servers, the botnet implementation over that kind of network structure can hide itself effectively. To analyze that kind of interaction, investigators need to solve all network elements in the TOR network. But the whole aim of a TOR network is to focused on preventing this; in short, this can be done by combining botnets and/or similar malicious organization channels to design a new type of TOR network. In this thesis, fully open source programs, codes, and explanations for the example installations of a similar

environment are provided. The chosen open-source tools and necessary codes are explained in detail in the APPENDIX part.

### **1.1. Scope**

The study implemented an internal network within the Yaşar Cyber Security Lab. The used tools were 1 Linksys home-type modem, 2 desktop PCs, 27 virtual network elements, and 10 Linux Containers (LXC). The application ran in a Linux environment.

All machines were connected to an internal network, and 27 virtual machines provided the necessary infrastructure for the TOR network. The number of relays has been increased to allow extra circuitry. One of the computers had 27 virtual TOR elements, the other computer participated in this network as a bot master and contained the LXC system, which served as 10 bots. The network structure was established to perform thesis studies, and can work with the same function when connected to the internet.

### **1.2. Motivation and Aim**

The purpose of this thesis is to raise awareness of how specialized TOR networks can play a role as a game changer. The application of a botnet was determined only as an example. This environment not only provided a botnet communication medium, but it also provided a playground for intelligence and hacktivist groups. It is necessary to analyze all the anonymity provided by the TOR network in order to carry out any analyses and research in this area.

### **1.3. This Work's Novelty**

This thesis provides a novel idea to anonymize communications in a similar fashion to the already created TOR structure. Our latest research attempted to add a new layer to the onion structure of the network to increase its provided secrecy and anonymity. To the best of my knowledge, there are no such studies about this kind of possible implementations. This article is trying to create an awareness about the depth of this topic.

### **1.4. Literature Review**

The presence of botnets over TOR networks was first announced in DefCon18 (Brown, 2010). The concept basically TOR-ified the communication between the Command

and Control (C&C) server and bots. 2 years after the presentation, the first known botnet over TOR was analyzed by (Guarnieri, 2012), as announced on Reddit. The analyzed botnet was the modified version of Zeus with DDoS, with bitcoin mining and data theft capabilities. The bot master published, “Everything operating tru TOR is in a hidden service so no feds will take my servers down.”

Another situation was mentioned in 2013 in the TOR mailing lists (Dingledine, 2013). In a short period of time, there was a huge amount of network usage. At the first encounter, it was thought to be a normal condition (Munson, 2013), but researchers found that a huge botnet switched all communications to TOR network (Yklijnsma, 2013).

To the best of our knowledge, in the literature, the botnet over TOR designs does not involve creating another layer for a hidden network. Therefore, with this peculiarity, this research is novel. Since there are no other published works that combine botnets and TOR structures in such a fashion, our literary review is given separately. The research of a botnet within a TOR is indicated at the end of the literary review section.

#### **1.4.1. Researches Based on Botnets**

(Silva, Silva, Pinto, & Salles, 2013) In the survey, detailed information about botnets' handling, development patterns, development aims, architects, and behaviors are available. It is stated that there are approximately 16-25% computer boots connected to the internet. In the survey, the money flow that the botnets caused was found by the FBI. In 2006, the estimated amount of financial loss due to malware was \$13.2 billion. There is a detailed taxonomy about botnets that has been kept until 2011. Survey De focused primarily on botnet detection and defense techniques. A detailed description of botnet architectures, behaviors, and detection and defense techniques is available in the background section.

(Rodríguez-Gómez, Maciá-Fernández, & García-Teodoro, 2013) This survey has been done in recent years to overcome problems arising from an increasing amount of research on botnets. The surveys differed from the previous surveys because they analyzed the botnets by fitting them into a life-cycle model of their own. A botnet's life-cycle consisted of its conception, recruitment, interaction, marketing, attack execution and success sections. The stages in this model are subdivided among themselves. The subheadings of the conception part include its founding motivation, design, and implementation. Subheadings in the interaction phase include “internal”

and “external”. Botnets' hidden mechanisms are also mentioned in the survey. There is a detailed taxonomy that has been devised according to the life-cycle model. This research has not addressed the botnet detection, defense or prevention mechanisms. The survey focused on the architectural design and behaviors of botnets.

One of the new trends, using botnet C&C centers over social networks, is examined in the report by (Ave, Pantic, & Ave, 2015). The aim of the paper was a stenographic system that demonstrates the feasibility of the social networking website Twitter as a botnet C&C center that an attacker could use to reliably communicate messages to a botnet. In that approach, Twitter account names are automatically generated based on Markov chains so that the bots can connect new accounts if the existing bot master account is unavailable. In social networks, millions of users interact and communicate in real time. By using this information, these networks are well suited for crafting cyber-attacks, such as in controlling an existing botnet. Also there are analyses about Domain Generation Algorithm (DGA)-based botnets detected over social networks (T.-S. Wang, Lin, & Lin, 2016). In the correlated paper, efforts are made to detect DGA-type botnets by analyzing different patterns of DNS queries to claim that DGA botnets have much more DNS traffic than with normal users by examining stage name error responses, called NXDOMAIN responses. Then, the researchers try to reach a methodological goal regarding group hosts, and determine if it's compromised or not. Filtering and clustering modules were used as a method for caching that information. Botnets find it easy to count on PAT attacks to complete their objectives (Thakar, 2016), in which the Persistent, Advanced Threat attacks become easily deployable. The decreased cost of technology allows adversaries to implement those kinds of attacks more frequently without needing governmental-level economic support. Botnets are the main tool for those Persistent, Advanced Threat (PAT) attacks. Powerful botnets receive the economic, intelligence, and computation power needs for PAT attacks. In that research, botnet families were analyzed in detail and the properties of botnets were explained. Centralized and decentralized botnet methodologies, fast-flux networks, and the stages of forming a botnet were examined. A detailed botnet timeline matrix, botnet infection distribution by countries, and a source type for bots can be found graphically in the research.

Scaling, detecting and preventing further research on botnets are mostly studied for the thesis (Chang, Mohaisen, Wang, & Chen, 2015). In that topic's research, the collected data showed the unique IPs of botnet families. The investigation tried to

analyze the characteristics of botnets and show important collaborative works between different botnets. The Team Cymru Community formed the data from monitored log dumps of 207 days total. The specified parameters were the botnet size, its stability, and its elasticity. The size of each botnet was found by counting unique Internet Protocols (IPs). The stability was decided by the active duration of each botnet family. Elasticity was defined as the ratio of the maximum number over the minimum number of live botnets. One of the important outcomes of the paper was defining the collaboration between botnet families, which occurred when some of the bots were used by multiple different C&C servers. The research about the characterization of a C&C in large-scale networks (Fontugne & Mazel, 2016) analyzed 5 months' traffic of different botnet families, and tried to characterize 6 different C&C roles. The mentioned life cycle stages of botnets were conception, recruitment, interaction, and attack. Analyzed data sets have two meanings in research, first they identified botnets and C&C servers by using blacklists. Second, they tried to extract botnet behaviors from the collected data. For the examinations, the researchers used 126 days worth of collected data and more than 14 TiB. The roles of C&C servers were identified by two parameters, the average packet size and the traffic direction.

The given parameters:

- A Scan: The largest number of packets are evaluated. The packets are only sent by C&C servers.
- Keep-alive: There is an equal quantity of sent traffic from servers and bots. This is the typical signaling of a Heartbeat mechanism.
- Interaction: This packet-carrying payload is around 100 bytes and equally sent between servers and peers, usually over Secure Shell (SSH) or Hyper Text Transfer Protocol (HTTP) ports.
- Pull: These packets have the highest amount of data sent by peers to the C&C servers. This is the sensitive data of infected hosts, and the traffic is over HTTPS.
- Push: These packets are sent from the server to peers in different quantities. This role can be the recruitment and interaction stages in the botnet life-cycle.
- Mix: This includes various types of traffic, and seems like a mixture of the different roles.

Another research about the large-scale analyses for botnet detection was published by (Bottazzi, Italiano, & Rutigliano, 2016). Periodic behavioral patterns of infected hosts communicating with their C&C servers are addressed in this paper. Botnets are typically propagated through malware. The main part of the research is an analysis done with proxy logs instead of a Fast Fourier Transform. Therefore, an analysis using the proxy logs method can be easily implemented in any real-world scenario, and decrease memory and computational cost. In a frequency domain analysis, continuous signals are representative. The set of these frequencies is defined as the frequency spectrum of the signal. A frequency domain analysis allows the characteristics of domains to be found out.

Some features used to extract valuable information in this research are:

- The average duration of connections.
- The average number of source bytes and destination bytes per flow.

Each of these attributes are stored as proxy logs, and as an experimental setup in which researchers use a network of 60,000 workstations with over 100,000 users to analyze the information. The total size of an analyzed file is more than 100GB. At the end of the analysis, two malicious servers are found and categorized.

The analysis of botnet detection and prevention mechanisms are also studied in this thesis (Haddadi & Phan, 2016). Whichever approach is used, the evolving nature of botnets and the required pre-defined botnet detection rule sets which were employed may affect the performance of a detection system. In this paper, the effectiveness of two rule-based systems and two machine learning-based techniques is compared, and thirteen public botnet data sets are used. After botnets started using more common protocols, decentralized topologies and encryption/detection became more challenging. Different kind of detection systems have been developed.

Detection features mentioned in the research are listed below:

- a Packet Payload system / Machine learning
- a Flow based system / Machine learning
- Snort /Rule-based detection
- the BotHunter botnet detection method / rule-based detection

Also, proofs of the concepts behind botnet detection systems are studied (T. Wang et al., 2016). DGA is a technique that allows botnets to evade detection systems. Domain

generation algorithms (DGA) are frequently changing the listed domain while C&C communicates with bots. The DNS traffic-based detection systems' technique is becoming ineffective. In this paper, the complex relationships between botnet populations can be possible to analyze with a new taxonomy of DGAs which captures their characteristic DNS dynamics. Botnet detection researches also can be performed regarding a mobile domain (Tansettanakorn, Thongprasit, Thamkongka, & Visoottiviseth, 2016). like with ABIS, the Android Botnet Identification System prototype, which is a developed system check application, and has an Android Package Kit (APK) file which determines if there is possible malware or not. Machine-learning approaches are also implemented in the ABIS for those determinations, which are made via two methods of detection: static and dynamic. Static detection tries to constitute all of the botnets' signatures by comparing their pattern characteristics and binary files, and researchers realized that their API calls and permissions are good bases for classification. Dynamic detection, mainly relies on Honeypot and HoneyNet kinds of security approaches regarding output data. The data was studied and published by a university, and the list contains 1,929 categorized Android botnet APK files. Such researchers use machine-learning tools like Weka for getting results. For the reverse engineering steps of collected android programs (APK programs), APKtool has been used. Trained data sets only detect and react to Android botnet malwares. ABIS has a high success rate in known botnet families.

In this paper (Al-Jarrah et al., 2015) , researchers proposed a state of the art traffic based Intrusion Detection System (T-IDS) built on a novel Randomized Data Partitioned Learning Model (RDPLM), relying on a compact network feature set and feature selection techniques, simplified subspace and a multiple randomized meta-learning technique.

Methodologies used in this research:

- Collecting and generating botnet data sets which represent real-world traffic
- Developing an efficient and scalable ML algorithm that can dealt with the large-scale network traffic and provide real-time detection within qualified real-time
- Developing a novel feature selection technique that eliminates redundant and irrelevant data
- A meta-learning model of multiple randomized trees, which considers a randomly selected feature's subset with no pruning, is constructed to detect botnet attacks

Some of the state-of-the-art researches about the misuse of protocols (Schales et al., 2016) state that a DNS provides a highly attractive channel for advanced threat actors and botnet operators to establish hard-to-block and stealthy communication channels between infected devices on C&C infrastructures. The Fast-fluxing (FF) and a domain generation algorithm (DGA) are two well-known public DNS exploitation techniques to build agile C&C mediums. A Fast-Fluxing domain name technique is based on poisoning the DNS server, and frequently changing its domain name and resealed IP pair. With that technique, common configured firewall systems are dodged easily. Detection with these two underlying techniques is a big data problem. In that paper, MapReduce and Feature collection and correlation Engine algorithms were implemented as solutions for DGA and FF problems. In the paper by (Andriesse, Rossow, & Bos, 2015), decentralized peer-to-peer (P2P)-type botnets were analyzed in detail. Common detection techniques were useless towards decentralized botnets. That research claimed that knowledge about the nodes in a botnet and connections between them is obtained using intelligence-gathering techniques. Malware analysts have used recon tools to monitor high-profile botnets, and they suffer from serious protocol deficiencies, making them easily detectable. Reconnaissance methods for P2P botnets can be divided two classes: crawler-based and sensor-based methods. Also, anti-recon attacks, recon anomalies, and stealth versions of the crawling technique have been examined in detail.

#### **1.4.2. Researches Based on Tor network**

(Dingledine, Mathewson, & Syverson, 2004) Research defines all necessary data structures, network architectures and applied cryptographies. Basically, the related study introduces how a TOR network works and how it has been improved for more efficient anonymous communications. The research also mentioned weak parts of TOR networks. The TOR's background was touched on with detailed information. Some of the studies mentioned in this thesis concern how state-of-the-art improvements can be implementable on TOR networks. In the research by (Kale & Wu, 2016). one of the most prevalent drawbacks of Onion Routing is the unfair distribution of traffic, which they were attempting to resolve. The researchers were implementing a circuit switching model on modified entry onion routers with an experimental setup. In the results, onion proxies achieved better traffic distributions



without reducing the throughputs. This approach allowed the entry onion routers to quickly detect congestion and enable both light and bulk traffics to achieve a better performance. Also, (Syverson & Boyce, 2016) research focused on how onion network domains can avoid trust issues by using certificate systems; 3rd party trust mechanisms are against the nature of TOR networks. Nevertheless, certification or any kind of trust and privacy mechanisms can be implemented into a TOR network. The certificate systems' availability was discussed in that research. Detailed examinations about a TOR network and the certification system for web authentication can be found.

Regarding different aspects, anonymity is also looked at in one of the papers, which was composed by (Tsudik, Uzun, & Wood, 2016). It proposed communication which relies on a Content-Centric Networking (CCN). Previous works have examined ANDaNA and compared it in detail with a new version of it, ACCCN. There are two noteworthy characteristics of a CCN. First, the content is named, addressable, and routable in a network. Second, all content is signed for by its producer. These characteristics of the CCN compose the key point of that research. The proposed ACCCN design is parallel to a TOR network in its onion-style encryption and decryption. Researchers claim that ACCCN can support high-throughput, low-latency, unidirectional and bidirectional traffic over CCN, with anonymity guarantees equivalent to a TOR network. Also, this research is in the same domain (Burgstaller, Derler, Kern, Schanner, & Reiter, 2016), and every reveal of a packets user contains sensitive information in terms of public IP address information. That research focused on how improvements can be made in terms of anonymity within an onion routing system. The compartmentalization of anonymity frameworks is found in the Background and Related Works section. Highly detailed research was done about how onion routing works, especially about the layers of an onion routing system (including the communication layer, onion layer, and end-to-end communication layer). Researchers completed the methodology with security and performance analyses.

One study tried to provide a solution for anonymous network usage in conflict situations (Springall, De Vito, & Huang, 2015). The paper focused on how governments effectively manage Internet traffic during crises. Internet protocols haven't been designed for anonymity, privacy and those kinds of crisis situations. Onion routing systems can solve this problem with its design. In the paper, general mechanisms were proposed for constructing an onion routing layout over distributed and decentralized networks, including network structures which were organized

according to a peer-to-peer paradigm. They also presented the limitations that the original design of an onion routing system (including its most common implementation, TOR) has when it is used on a distributed network.

TOR networks are called “blackbox networks” because of their anonymous nature, which some don't always find to be favorable. Some of the researches examined here try to remove the anonymity of the TOR networks. In (Koch, Golling, & Rodosek, 2016) paper, the exit nodes of a TOR network are examined in detail. The results were compared using the statistical outcome of countries. Also, ISPs and governmental behaviors have been compared using the traffic results of TOR network's exit nodes. A detailed survey about TOR networks and their exit node selections are found in that paper.

Another paper, by (Springall et al., 2015), mentioned the attempted separation of TOR network connections and clear Internet connections from each other in order to detect the TOR network connections. The researchers determined the thresholds in three-way hand shake time. A simulated real-world traffic and eight participants were used in testing. A 100% success rate in HTTP protocols and a 99% success rate in SSH protocols have been confirmed in the analysis' results.

Implemented live botnets have been using the power of anonymity provided by a TOR network, and it seems that this approach will continue to exist in the future. In the report by (Sanatinia & Noubir, 2015), analyses, mitigations, and detections of botnets' activities over TOR networks were discussed. In that research, when a hidden botnet was detected over a TOR network, counter bots were created to attack the hidden structure which tracked down malicious activities. Another study (Casenove & Miraglia, 2014) focused on HS addresses which were compromised due to lack of design in the first steps of the botnet's life cycle, which caused vulnerabilities. When the malware exploited a specified computer, it must have been communicating with the C&C server in order to download its initial configurations. In the study by (Ghafir, Svoboda, & Prenosil, 2014), a campus network which was designed as a TOR network was studied. That research aimed to capture the anonymous network traffic using one of the fundamental weaknesses of a TOR structure by taking over all the DSs to identify all of its anonymous traffic. In using this specified technique, it's possible to avoid anonymized network activities, however, there is also the certain need to attack all captured circuits in the onion traffic with an offline dictionary-style decryption procedure to acquire necessary information.

## **CHAPTER TWO**

### **BACKGROUND**

#### **2.1. Botnet**

##### **2.1.3. Definition and Anatomy of a Botnet**

The name “bot” originated from the Slavic language, wherein the word “robota” is the name for a forced laborer. An internet bot, or simply a bot, is an application which is created by a software that runs automated tasks over the internet. In other words, bots are malicious codes running on a host computer which allow botmasters to control the host computers remotely. These kinds of malicious codes enable botmasters to perform various actions. A botnet is a customized network created by the bots connected to the internet. A whole botnet is controlled by its botmaster (attacker) through a Command and Control (C&C) server. There are many possible configurations for developing, establishing, controlling and managing the botnet. Whatever these specialties are, bots, a C&C server, and a botmaster are always included.

Botnets are primarily used in fraudulent online activity committed by criminals or hackers. With mass reproductive and auto-delivery mechanisms, botnets can be multiplied in an endless way. It is very likely that there are more uncharted bots within known botnets. Botnets can be utilized for a variety of interruptive activities, performing Denial of Service (DoS) / Distributed Denial of Service (DDoS) attacks, information harvesting, identity theft, and spamming and manipulating online games. An increasing number of botnets makes them an extremely destructive weapon on the internet. Almost every machine that has a Network Interface Controller (NIC) device can be included in a botnet. In the most recent cases, it has been seen that some Internet of Things (IoT) devices are included in botnets. Adversary botnet developers are able to do even more when IoT devices are included, and the effectiveness of botnets are highly increased when that method is added.

An Internet Relay Chat (IRC) is a text-based chat system which takes shape in different channels. Users who want to chat can start communicating on the IRC system simply by connecting an IRC server and selecting a channel. Administrators of the channel can control users by pre-defined channel rules with little helper scripts. These scripts are the real source of botnets. Eggdrop was the first IRC bot which provided automated

administrator support to make things easier. Eggdrop was released in 1993 (Fisher, 1998). Not long after, malicious versions of IRC bots appeared that were developed by adopting the same basic idea. However, they focused on DoS and DDoS type attacks. Botnets progressed rapidly to create the next versions.

Spreading, attacking, and communicating mechanisms are becoming highly robust and sophisticated. Although, the specific components' approaches are still the same (bot, botmaster, and the C&C server).

### **Life-Cycle of Botnets**

In order for an affected host to be an active bot and be a part of the botnet, the host must pass a life-cycle that integrates the information (Silva et al., 2013)(Rodríguez-Gómez et al., 2013). The phases of botnet formation sometimes were identified by different names, but they have similar meanings at the end.

Conception: This is the most important phase of creating a botnet. The real causes underlying the establishment of a botnet need to be understood. Then, the architecture and design are formed according to the real cause. The conception phase can be divided into three processes: motivation, design, and implementation.

Motivation: The main reason a botmaster creates a botnet is their motivation. It's generally classified as money, entertainment, ego, status or entrance to social groups. There is a detailed list of prices for botnet services in the necessary reference (Xv, April, Fossi, & Johnson, 2010). Potential botnet infections, communication types, and overall architectural structure are decided at this stage.

Design: There are several different kinds of design; these are centralized, distributed, and hybrid. Bots directly communicate with their botmaster in centralized botnets. All botnet are controlled from one central C&C server. That system can develop weaknesses due to mistakes in these parts. Adoptable botnets can tolerate those kinds of problems using several techniques, like DGA (DGA is explained in detail in "Botnet Evasion Systems."). In a distributed architecture, all the bots act like a C&C server and control each other. These kinds of botnets are more durable than centralized botnets. There are some tradeoff communications, and the bots' spreading takes much more time than in a centralized version. Hybrid botnets use the advantages of both centralized and distributed (decentralized) botnets. There are distributed parts in the

botnet, but all the midpoints are connected to the C&C server directly. When one or two of the distributed parts collapse, other portions of the botnet keep standing.

Implementation: This sub-stage is just the implementation of an architecture. It's the software development process with respect to a specified design and motivation.

Recruitment: Also, known as infection. This stage involves implementing a botnet in a real environment. Bots must be recruited, and the botmasters' aim is to recruit as many bots as possible. This stage is unique in every botnet. In order to increase its capabilities, botnet can use zero-day exploits, scamming, and social engineering techniques all together.

Interaction: This phase defines all of the interactions between the bots, botmaster and the C&C server during the botnet operation. The interaction stage can be examined in its several sub-topics. External communication channels have a botmaster monitoring the botnet's behaviors, and the botnet and external server communications can be maintained from there. Internal communication channels handle information flow between the bots and the C&C server. Internal interactions can be studied in two sub topics, registration processes and C&C communications. Registration is the initial infection phase, which has to be done from another channel, much like ciphering. Once the registration is done, C&C server communications can start and a bot is ready to interact with the C&C server.

Marketing: Marketing is the final stage of the botnet life-cycle model. It is the time when the bot network reaches a substantial surface and harvests the goods from the network. In this stage, the developed botnet is marketed as a service or as a do-it-yourself (DIY) type of source code. Renting a service from a botnet works like renting a domain name from a provider, but only the rented service can execute DDoS attacks or provide anonymity (Xv et al., 2010). However, interestingly, DIY kits are available as an open source code. These aim to hide the actual developer by revealing the source code and making it public. This is controversially opposed to botnets that provide anonymity by hiding the information, since in those the source code provides anonymity by becoming public.

### **Botnet Evasion Systems**

Evasion systems are also considered part of the proposed botnet life-cycle. These mechanisms are designed to hide the botnet and make it harder to realize and

investigate it. These hiding mechanisms can be categorized again as host-based and network-based. Common host-based techniques are binary obfuscation, versioning, and IP and email spoofing.

Binary obfuscation: This technique is the effective anti-reverse engineering technique applied on the source codes of bots.

Versioning: Having bots' source codes within modules and switching between them when IDS or anti-virus programs are triggered is known as polymorphism.

IP and Email spoofing: Mechanisms are also used for mystification against detection systems.

Some contemporarily known network-based hiding techniques are multi-hopping, ciphering, DGA and fast-flux networks, and anonymization.

Multi-hopping: This technique is a networking mechanism designed to make it difficult to track the source of an attack or C&C server. When based on the C&C server or the botnet's operator, multiple proxies are used before reaching the bot or any interface of botnet hierarchy.

Ciphering relies on the power of cryptography. In this strategy, the communication channel between bots and C&C servers are encrypted, and even if the botnet has been compromised, the encrypted packages need to be cracked.

Domain name generation (DGA) is yet another technique that improves the covertness of the botnet. In the mechanism, randomly generated domain names are used as meeting points. The number of generated domains can reach up to tens of thousands. This includes potential rendezvous points, which can disable blacklisting mechanisms.

Fast-flux networks: These attach hundreds to thousands of IP addresses onto a fully qualified domain name and switch them in a pre-defined frequency. A website that switches between a set of frequencies in the fast-flux network is much more than a domain, though. The domain becomes a nice hiding corner for the C&C server, and the frequency changes protect the actual identity of the user.

Anonymization: Taking advantage of the anonymity provided by TOR networks is a new trend among botnet developers. The first known implementation of such a botnet was in 2010. It was a modified version of the Zeus botnet with extra capabilities for anonymity (Falliere & Chien, 2009), and it had been published through Reddit

(Casenove & Miraglia, 2014). The anonymity of TOR networks and the dangers of a botnet have been combined since then.

#### **2.1.4. Detection & Defense**

Researches about detection and defenses studied in this thesis can be grouped in multiple categories: Honeynets, and signature-based and anomaly-based detection systems.

A Honeynet is a part of the Honeypot family, a term which originated from intelligent human techniques. This technique refers to a strategy where an attractive female or male spy takes advantage of vulnerable human victims, and creates and exploits their formed sexual relationship to force them to cooperate with them. In terms of luring and trapping, a similar technique is used in cyber space with the same terminology. A decoy-based information gathering system is prepared as a vulnerable member of a network. The “person's” vulnerabilities lure the attackers into traps. While the attackers exploit their victim, the Honeypot system analyzes and logs all of the behaviors of the adversaries. A Honeynet is used as an information gathering mechanism on a botnet all over the cyber domain. At the same time, this kind of mechanism has some problems, which were inherited from their core theory. The span of the botnet's discoverability and zero-day vulnerabilities are two limitations of Honeynets' detection systems.

Intrusion detection systems (IDS) are another important and functional tool against botnets. IDS mechanisms are based on two main defense techniques: signature-based detection systems and anomaly-based detection systems (Kugisaki, 2007). Signature-based detection systems are basically comparison tools for the defined peculiarities of known bots. These properties are determined by the monitored botnet traffic, the communication pattern, utilized ports, protocols and services. Although signature-based detection systems can rapidly detect known, analyzed bots, they can never be successful for even slightly different bots and vulnerabilities. Theoretically, the technique cannot detect a bot's activity before an attack occurs.

Anomaly-based detection systems are a favorite topic of study in botnet detection (Binkley, 2006). The technique is based on capturing suspicious activities within the network and the host. Any usage of pre-defined ports, services, and unusual network traffic latency can be counted as a suspicious activity. The main challenge about an

anomaly detection is defining what is normal and what is an anomaly. The method can detect new kind of botnets with a good configuration, and at that point, anomaly-based detection systems are separated in two main categories: host-based and network-based approaches. The anomaly-based detection technique involves a learning process where the anomaly is statistically defined. This process is also the weakness of the technique, as the open-sourced botnets are versioned on the dark market almost immediately, then the learning process becomes a nightmare. For these matters, security specialists use the hybrid approaches of all mechanisms.

## **2.2. Tor Network**

### **2.2.1. Anatomy & Definitions of Tor Network**

TOR networks originally showed up as a third-generation onion routing project in the U.S. Navy to protect governmental communications. Then, the TOR networks were publicized for worldwide usage. An increasing number of volunteers and participants increased its anonymity and efficiency. Anonymity based on cryptographic protocols was implemented in a multi-layered structure. Utilizing this type of structure, TOR layouts created a confidential communication medium between the senders and receivers.

The main components of a TOR network are listed below:

An Onion Proxy (OP): Defined as clients, it can be entry or exit nodes and handle the connections of the user.

An Onion Router (OR): Nodes are used to manage and establish a TOR network here. They can form a relay, a hidden service or a directory server.

A Hidden Service (HS): Anonymous public services which are allowed by a TOR network.

A Hidden Service Directory (HSD): In this component, the HS uploads a description of itself to initialize communication with clients.

A TOR network runs as a computer processes. The user-level processes can be configured as an OP or OR on every installed node. An installed TOR network starts as an OP in default. Users need to reconfigure it if they want to change computers for another kind of TOR network component. OR/OP machines communicate from



specified ports with a specified data structure. TOR networks originally used SOCKS ports and TLS connections while establishing communication mediums (IETF®, 1996)(IETF®, 2008). The communication between the nodes of each network was made with size-fixed cells and organized circuits. TOR networks' traffic stands on the specified cells defined by the configuration. The cells are examined in two subcategories: control cells and relay cells. The header of the circuit identifier specifies whether it is a control or a relay cell. Control cells manage the communications between OP/OR and OR/OR, as well as relay cells which control end-to-end streamed data with specified commands.

Creating a circuit with source and destination cells is the beginning of the process. The source then starts to construct a circuit with a symmetric key, and at every step (hop) of the way, it adds another symmetric key and encryption. Once the circuit has reached its destination, a message containing half of the Diffie-Hellman handshake (IETF®, 1999), which was carried by the circuit, is delivered. Then a connection is established with responses from the destination.

TOR network structures use RP to connect the clients and services at a certain level of anonymity. Web services on a clear network turn into a HS (as well as an OR) in the TOR network. The structure of communication between a HS and OP is created by specifying introduction points and making advertisements on the nodes to advertise its public key. The HS then builds a circuit on each of the advertisement nodes and waits for requests. When the service recipient wants to connect to the HS, it chooses a RP to connect with and builds a circuit to that RP. When the RP is ready, a stream is opened for that advertisement point, and another RP is created by the user. Once the two RPs are linked and the stream information is transmitted, the HS builds a circuit to the new RP via the client, and creates a stream circuit. Then finally, nodes start to communicate anonymously.

These steps are mentioned in the original TOR Network data structure paper (Dingledine et al., 2004)

- 1. Bob generates a long-term public key pair to identify his service.*
- 2. Bob chooses some introduction points and advertises them on the lookup service, signing the advertisement with his public key. He can add more later.*
- 3. Bob builds a circuit to each of his introduction points, and tells them to wait*

*for requests.*

- 4. Alice learns about Bob's service out of band (from hidden-wiki kind hub web pages). She retrieves the details of Bob's service from the lookup service. If Alice wants to access Bob's service anonymously, she must connect to the lookup service via Tor.*
- 5. Alice chooses an OR as RP for her connection to bob's service. She builds a circuit to RP, and gives it randomly chosen "Rendezvous cookie" to recognize Bob.*
- 6. Alice opens an anonymous stream to one of Bob's introduction points, and gives it a message (encrypted with Bob's public key) telling it about herself, her RP and rendezvous cookie, and the start of Diffie-Hellman handshake, and a hash of the session key they now share. Alice knows she shares the key only with Bob.*
- 7. The RP connects Alice's circuit to Bob's Note that RP can't recognize Alice, Bob, or the data that transmit.*
- 8. Alice sends a relay begin cell along the circuit. It arrives at Bob's OP, which connects to Bob's webserver.*
- 9. An anonymous stream has been established, and Alice and bob communicate as normal.*

Another important point is that there is a need to increase the reliability of the system and sustainability of the relays. So, on the second-generation TOR networks, Directory Servers (DS) are generated. A group of reliable, well-known ORs are used for tracking changes in the network topology, including keys and exit policies. Directory Servers work as HTTP servers. Clients can easily get updated about the list of services. Other ORs can upload their states into that HTTP server as well; ORs in the network send updates of their signed information frequently to their directory servers. The directory servers process this information and store it as a list of the network nodes and their detailed states. A client, becoming an OR internal node, then pulls the list from the chosen directory server. Before the creation of second generation TOR networks, this list of details is flooded into the network by the ORs. Putting in the directory servers not only increased the efficiency, it has also increased the secrecy of the TOR networks.

The TOR network version used in this research is The Second-Generation Onion Network. This generation is the latest major updated version of TOR. There have been several minor upgrades in TOR network structure after that version. Most of them were associated with efficiency. Nevertheless, the TOR network still has some serious weaknesses. The next section is about to touch on these problems.

### **2.2.2. Vulnerabilities of Tor Network Anonymity**

In order to explain the work done in this thesis, the vulnerabilities of the TOR network will be grouped and presented as active, passive, directory attacks and attacks against rendezvous points. The known weaknesses of a TOR network are listed and discussed below.

#### **Passive attacks**

Observing user traffic patterns: Although the anonymity does not deteriorate, the traffic patterns of the users can be revealed. Different application streams may be executed over a single circuit, mapping the traffic patterns of users requires more processing power.

Observing user content: An encryption algorithm is applied to the user-side content, as defined in a TOR structure. However, the connections to responders are not obliged to be encrypted, thus a weakness is observed when the connected service is an adversary.

Options' distinguishability: Sometimes system admins can choose to be identified in order to increase the popularity of the web service. Configurations made in this direction can cause a loss of anonymity. The configuration of the TOR network can allow such chosen mistakes.

End-to-end timing correlation: When an attacker is watching patterns of traffic at the initiator, there is a high probability that the responder will be able to confirm the correspondence's origin. The solution is to hide the connection between the OP and the first TOR node. This method requires an observer to separate traffic originating at the OR from traffic passing through it.

Website fingerprinting: Rather than searching exit connections for timing and volume correlations, the adversary may build up a database of “fingerprints” containing necessary information between the clients and websites. It can be confirmed later on as information stored in the database (Hints, 2002).

## **Active attacks**

Compromise keys: An attacker learns that the TLS session key can see control cells and encrypted relay cells on every circuit on that connection. It is just a like peeling onion after learning the TLS session key. Periodic key rotation limits the window of opportunity for these attacks. On the other hand, attacker who learns a node's identity key can replace that node indefinitely by sending new forged descriptors to the directory servers.

Iterated compromise: An adversary who can compromise ORs by intrusion techniques could leak down the circuit compromising nodes until he reaches the end. The adversary has to complete this attack within the life time of the circuit.

Running a recipient: It is easier to execute ENT-to-end attacks. If the adversary can induce users to connect to his webserver, he now holds one end of their connection.

Running an OP: In some settings, it may be necessary for the proxy to run remotely. That may cause to monitor the activity of those connecting to proxy. End users can run their local OP easily.

DoS non-observed nodes: An observer who can only watch some of the TOR network can increase the value of this traffic by attacking non-observed nodes to shut them down.

Running a hostile OR: In addition to being a local observer, an isolated hostile node can create circuits through itself, or alter traffic patterns to affect traffic at other nodes. Nonetheless, a hostile node must be immediately adjacent to both endpoints to compromise the anonymity of a circuit. If an adversary can run multiple ORs, and can persuade the directory servers that those ORs are trustworthy and independent, then occasionally some user will choose one of those ORs for the start and another as the end of a circuit.

Tagging attacks: A hostile node could "tag" a cell by altering it. If the stream was, for example, an unencrypted request to a Web site, the garbled content coming out at the appropriate time would confirm the association.

Replacing the contents of unauthenticated protocols: When relaying an unauthenticated protocol like HTTP, a hostile exit node can impersonate the target server.

Replay attacks: Some anonymity protocols are vulnerable to replay attacks, but TOR ones are not. In TOR ones, replaying one side of a hand shake will result in a different negotiated session key, and so the rest of the recorded session can't be used.

Smear attacks: An attacker could use the TOR network for socially disapproved acts, in other words, to bring the network into disrepute and get its operators to shut it down. Exit policies reduce the possibilities for abuse, but ultimately the network requires volunteers who can tolerate some political heat.

Distributing hostile codes: An attacker could trick users into running a subverted TOR software that did not, in fact, anonymize their connections, or worse, could trick ORs into running weakened software that provided users with less anonymity.

### **Directory Attacks**

Destroying directory servers: If a few directory servers disappear, the others will still decide on a valid directory. So, as long as any one directory server remains in operation, it will still broadcast.

Subverting a directory server: By taking over a directory server, an attacker can partially influence the final directory. Since ORs are included or excluded by a majority vote, the corrupted directory can, at worst, cast a tie-breaking vote to decide whether to include marginal ORs.

Subverting the majority of directory servers: An adversary who controls more than half of the directory servers can include as many compromised ORs in the final directory as he/she wishes.

Encouraging directory server dissent: The directory agreement protocol assumes that directory server operators agree on a set of directory servers. An adversary who can persuade some of the directory server operators to distrust one to another, could split the quorum into mutually hostile groups, thus partitioning users based on the directory they use. A TOR system does not address this attack.

Tricking the directory servers into listing a hostile OR: A TOR threat model explicitly assumes that directory server operators will be able to filter out most of the hostile ORs.

Convincing the directories that a malfunctioning OR is working: In the current TOR system implementation, directory servers assume that an OR is running correctly if they can start a TLS connection to it. A hostile OR could easily subvert this test by accepting TLS connections from ORs but ignoring all cells. Directory servers must actively test ORs by building circuits and streams as appropriate.

### **Attacks against rendezvous points**

Making many introduction requests: An attacker could try to deny Bob service by flooding his introduction points with requests. Since the introduction points can block requests which lack authorization tokens, Bob can restrict the volume of requests he receives, or require a certain amount of computation for every request he receives.

An attack on an introduction point: An attacker could disrupt a location-hidden service by disabling its introduction points. But because a service's identity is attached to its public key, the service can simply re-advertise itself at a different introduction point. Advertisements can also be done secretly so that only high-priority clients know the address of Bob's introduction points or so that different clients know of different introduction points. This forces the attacker to disable all possible introduction points.

Compromising an introduction point: An attacker who controls Bob's introduction point can flood Bob with introduction requests, or prevent valid introduction requests from reaching him. Bob can notice a flood, and close the circuit. To notice a blocking of valid requests, however, he should periodically test the introduction point by sending rendezvous requests and making sure he receives them.

Compromising a rendezvous point: A rendezvous point is no more sensitive than any other OR on a circuit, since all data passing through the rendezvous is encrypted with a session key shared by Alice and Bob.

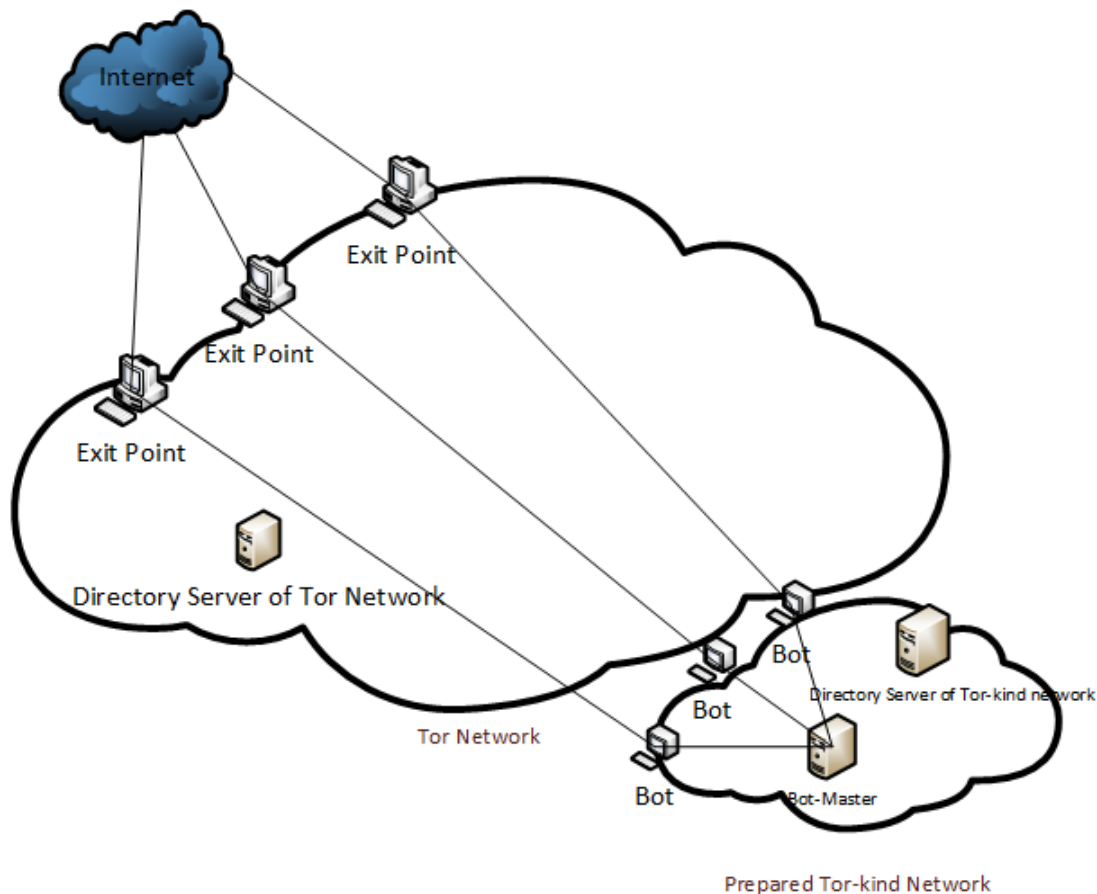
## **CHAPTER THREE**

### **IMPLEMENTATION**

#### **3.1. General Overview**

The main purpose of the thesis is to create different configured TOR networks and hide them near the original TOR network. Another crucial aim is to figure out whether TOR configurations may provide an extra layer of anonymity. In this chapter, the proof of this conceptual work is provided to show that it is possible to create undiscovered communications even from inside an anonymous network.

The core of this work is to configure directory servers on an isolated, configured TOR network, then connect that network to the original TOR one without connecting the directory servers. The configured TOR network and original TOR network can get in contact with entry and exit nodes. This situation ensures that a configured TOR network can connect to the Internet over the original TOR network without disclosing itself. Its general schema can be shown in **Figure 1**.



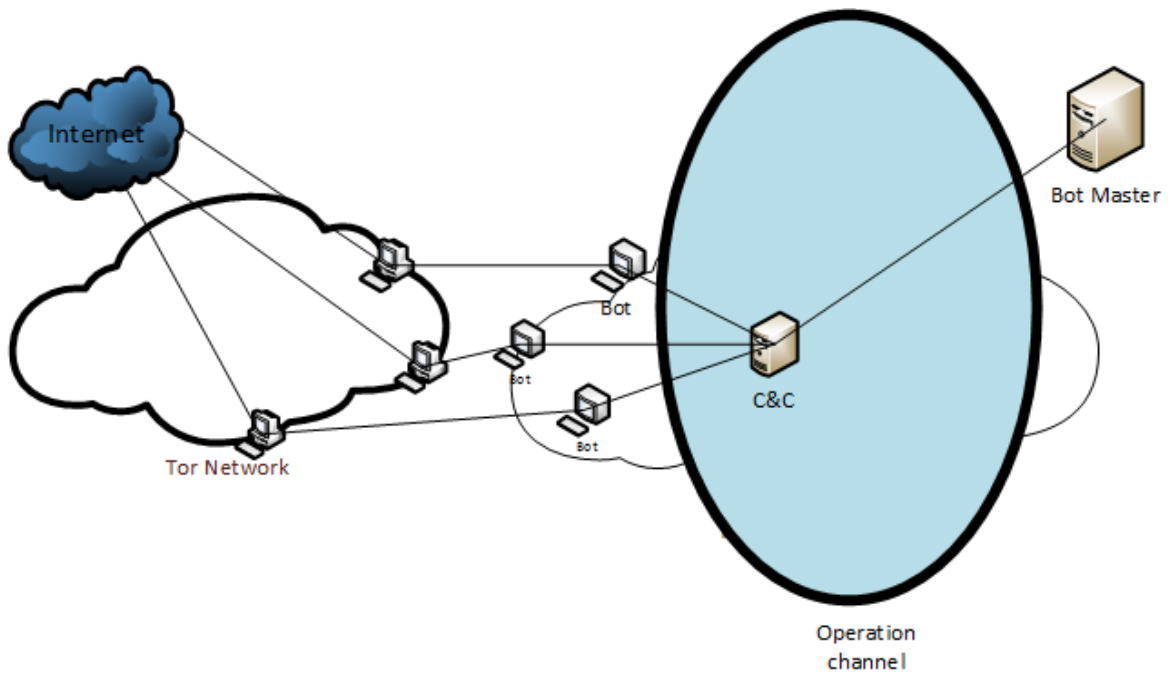
**Figure 1.** The General Network Schema

The botnet system has been added here to show an example usage of a configured TOR network. An anonymized usage of a botnet network results in only one of the endless possibilities of that configured TOR network.

The C&C activities of the botnet is anonymized by configuring a TOR network. The botnet architecture is used to increase intelligibility so the system is as simple as possible to use. The C&C server implemented with the TOR system is connected to an IRC server, and python-coded bots are added to the anonymized network. The communication between the bots and the C&C server is done via SOCKS ports (TOR-ified connections), so via a configured TOR network. The botnet recruitment process is out of the scope of this project. Bots communicate with a C&C server via a configured TOR network via a TOR-ified IRC server. The operation channel (communication between the botmaster, C&C server, and bots) of the botnet is anonymized by using the structure studied in this thesis with the approach above.

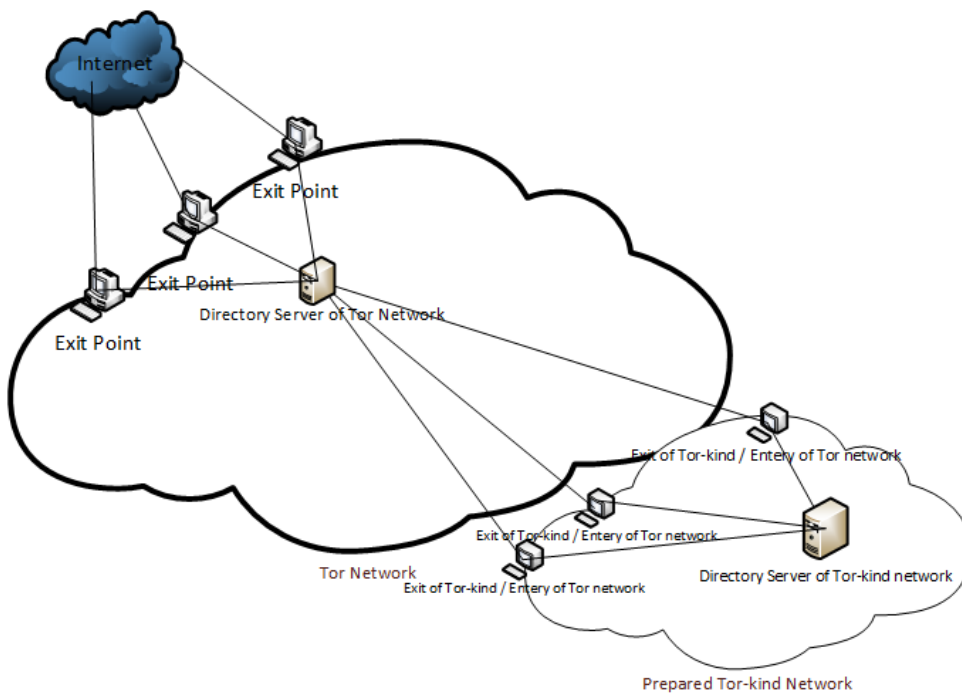
**Figure 2** shows the implemented approach of the botnet.





**Figure 2.** A Communication Operation Channel Schema

Regardless of what schemes are attached to the network, the anonymized structure ought to be added to a new TOR network to add an extra layer of anonymity, as provided with the original TOR network. As far as this proof of conceptual application is concerned, the botnet components are attached to the configured TOR network, as illustrated in **Figure 3**.



**Figure3.** The General System Schema

### **3.2. Architecture and Structure**

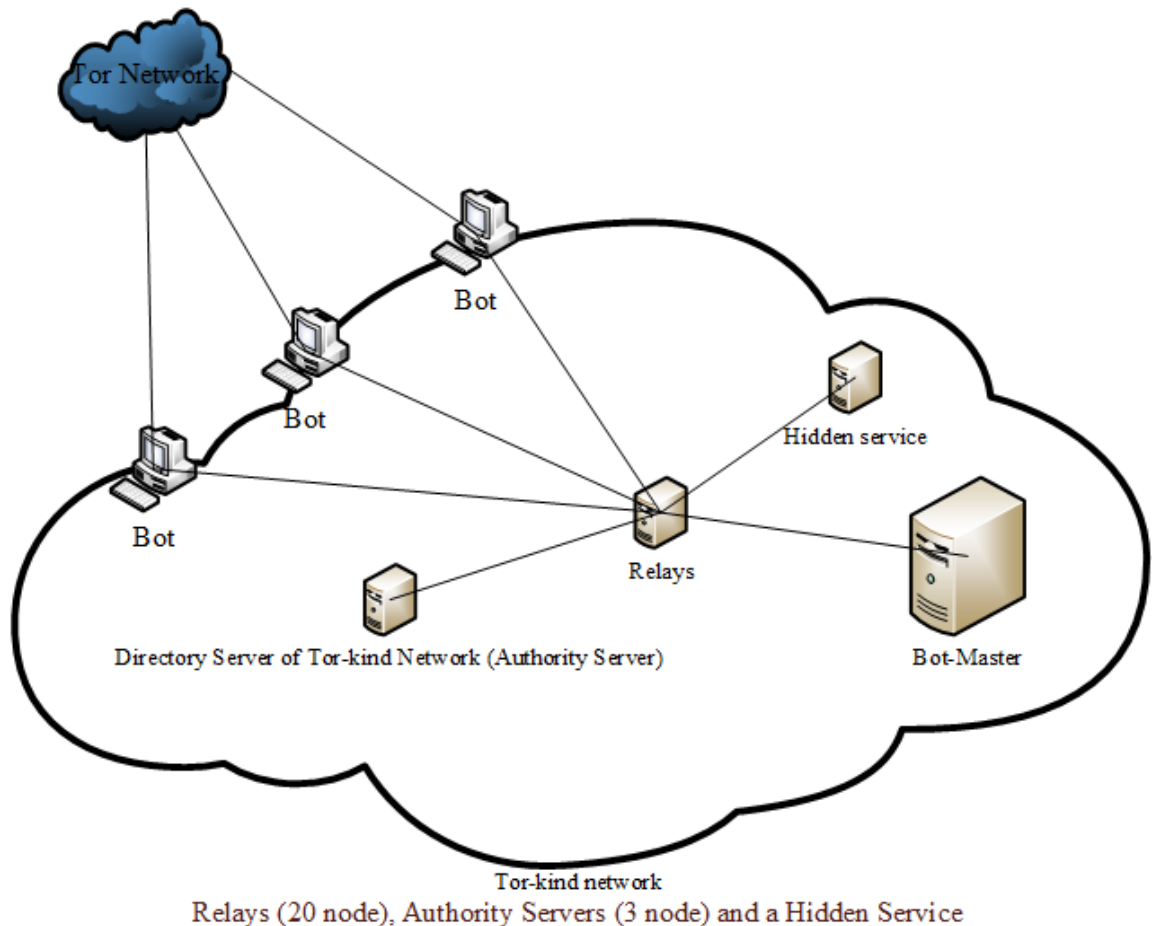
As they were described in the previous section, the necessary steps for installing the system are listed below:

1. Preparing a configured TOR network structure
2. Preparing a suitable botnet structure for the configured TOR network
3. Installing the system in the isolated lab and making it compatible with the network

An open source TOR network generator application (Chutney) was used, with major modifications, to create the necessary network structure. There are 27 modified TOR kinds of network elements, and 4 authority (directory) server nodes, 20 relay nodes, 3 client nodes, and 1 hidden service node was installed in a TOR kind of network. Authority servers, relays, and clients create the necessary structure of the TOR-similar network. A hidden service meets the requirements of an IRC server over the TOR-mimicking network.

A LXC system was created to hold bots. The bot-joining process was coded into the container as a system startup bash code. When the container starts, it automatically joins the modified TOR network, and then tackles the IRC server and joins the pre-defined channel. After the initial process is complete, the bots are ready for use. In accordance with the ethical rules, the implemented features of the bots were limited with example print functions. At the same time, joining the TOR network and using a corresponding IRC server has some real world examples. Details of an implemented system can be found in **Figure 4**.

All configurations and codes can be found in APPENDIX section.



**Figure 4.** Schema Details of an Implemented Botnet over a TOR System

The next section will touch the system's vulnerabilities and try to pass through them using possible solutions.

### 3.3. Vulnerability Assessment

The approach used in this thesis is meant to change the TOR network's configuration file. All weaknesses of the TOR network [which were defined in the study by (Dingledine et al., 2004)] were valid for the approach applied in this study.

The created system for this thesis, had a vulnerability that was inflicted from the botnet architecture. It was an exploitable part in the end node of the operation channel showed in **Figure 2**. The part may have caused a leakage for a configured TOR network's exit relay communication, or for an entrance relay within the original TOR network.

If the main study in this thesis is practiced in real life, there would be massive amounts of problems in it. Potential problems when using the system may show incongruences

with 3 of its basic principles of security. In order to clarify this, the thesis topic should be examined from the perspectives regarding privacy, authenticity, and assurance of integrity separately.

- From the privacy viewpoint, certainly the design of this system has been improved to keep the communicating parties anonymous. The power of the TOR network ensures that the authorities of the parties are kept secret. So, numerous methods are used to provide anonymity and privacy protection for the users.
- From the authenticity viewpoint, no study has been conducted on the reliability of the information transferred thus far. If the system is to be used in real life, system-specific reliability protocols should be developed. Besides, the TOR network did not seem to rely on authenticity. On the TOR network, the source and destination may not be what they look like. For example, the person who seems to be a drug dealer could be a CIA agent. This situation means that authenticity can be violated. If one wants to use this thesis' mentioned system in real life, system-specific reliability protocols need to be developed.
- At first glance, the cryptographic algorithms maintain the assurance of integrity. However, the use of different cryptographic algorithms may lead to other vulnerabilities. This thesis has not worked on this aspect due to its focus.

To keep the privacy of the working system at it's highest level, there is a need for virtual systems which are independent from the physical location. In case of any compromises, a virtual machine that would host a node of the hidden TOR network should be wiped out. A recent copy of the same structure can rebuild it while changing the identification and communication details afterwards. Indeed, to keep privacy at it's highest level, this circulation of virtual machines is compulsory.

## **CHAPTER SIX**

### **DISCUSSION AND CONCLUSION**

This thesis tried to improve and figure out a covered anonymized communication medium with a developed botnet mechanism. The developed system model can be used by governments, intelligence services and users whose identities should be kept secret. The implemented communication mechanism is well suited for botnet operations. The provided implementation in this thesis focused more on botnet activities and attack mechanisms. The main important result of this structure was to create a cyber-intelligence network where the information flow policies are on parallel cyber domains.

To accomplish this research, the TOR network data structure and anonymization principals have been studied in detail. This thesis' study continued examining the pearls and pitfalls of the TOR network. As a conclusion, this thesis provided a novel idea to anonymize communications in a parallel fashion to the created TOR structure.

The main statement of this study was a communication medium which was designed to conceal itself by utilizing a TOR network. The design provided an environment which enabled it to hide itself from other TOR networks by using an altered version. The number of such possible networks within an original TOR network can be increased. On the other hand, no studies have been found which find networks constructed with such configurations. Therefore, even if a complete TOR architecture is compromised one day, these parallel configured networks can still function under the radar.

## REFERENCES

- Al-Jarrah, O. Y., Alhussein, O., Yoo, P. D., Muhaidat, S., Taha, K., & Kim, K. (2015). Data Randomization and Cluster-Based Partitioning for Botnet Intrusion Detection. *IEEE Transactions on Cybernetics*, 46(8), 1796–1806. <https://doi.org/10.1109/TCYB.2015.2490802>
- Andriessse, D., Rossow, C., & Bos, H. (2015). Reliable Recon in Adversarial Peer-to-Peer Botnets. *Proceedings of the 2015 ACM Conference on Internet Measurement Conference - IMC '15*, 129–140. <https://doi.org/10.1145/2815675.2815682>
- Ave, W. T., Pantic, N., & Ave, W. T. (2015). Covert Botnet Command and Control Using Twitter. *ACSAC*, 171–180. <https://doi.org/10.1145/2818000.2818047>
- Binkley, J. R. (2006). An Algorithm for Anomaly-based Botnet Detection, 43–48.
- Bottazzi, G., Italiano, G. F., & Rutigliano, G. G. (2016). Frequency Domain Analysis of Large-Scale Proxy Logs for Botnet Traffic Detection, 76–80. <https://doi.org/10.1145/2947626.2947634>
- Brown, D. (2010). Resilient Botnet Command and Control with Tor. Retrieved from <https://www.defcon.org/images/defcon-18/dc-18-presentations/D.Brown/DEFCON-18-Brown-TorCnC.pdf>
- Burgstaller, F., Derler, A., Kern, S., Schanner, G., & Reiter, A. (2016). Anonymous Communication in the Browser via Onion-Routing. In *Proceedings - 2015 10th International Conference on P2P, Parallel, Grid, Cloud and Internet Computing, 3PGCIC 2015* (pp. 260–267). <https://doi.org/10.1109/3PGCIC.2015.22>
- Casenove, M., & Miraglia, A. (2014). Botnet over tor: The illusion of hiding. *International Conference on Cyber Conflict, CYCON*, 273–282. <https://doi.org/10.1109/CYCON.2014.6916408>
- Chang, W., Mohaisen, A., Wang, A., & Chen, S. (2015). Measuring Botnets in the Wild. *Proceedings of the 10th ACM Symposium on Information, Computer and Communications Security - ASIA CCS '15*, 645–650. <https://doi.org/10.1145/2714576.2714637>
- Dingledine, R. (2013). Many more Tor users in the past week? Retrieved from <https://lists.torproject.org/pipermail/tor-talk/2013-August/029582.html>
- Dingledine, R., Mathewson, N., & Syverson, P. (2004). Tor: The second-generation onion router. *SSYM'04 Proceedings of the 13th Conference on USENIX Security Symposium*, 13, 21. <https://doi.org/10.1.1.4.6896>
- Falliere, N., & Chien, E. (2009). *Zeus : King of the Bots*. Retrieved from [https://www.symantec.com/content/en/us/enterprise/media/security\\_response/whitepapers/zeus\\_king\\_of\\_bots.pdf](https://www.symantec.com/content/en/us/enterprise/media/security_response/whitepapers/zeus_king_of_bots.pdf)
- Fisher, J. “guppy.” (1998). Eggdrop. Retrieved from <https://www.eggheads.org/>
- Fontugne, R., & Mazel, J. (2016). Characterizing Roles and Spatio-Temporal Relations of C & C Servers in Large-Scale Networks. *WTMC*, (4), 12–23. <https://doi.org/10.1145/2903185.2903192>
- Ghafir, I., Svoboda, J., & Prenosil, V. (2014). Tor-Based Malware and Tor Connection Detection. In *International Conference on Frontiers of Communications, Networks and Applications (ICFCNA)* (pp. 1–6).
- Guarnieri, C. (2012). Skynet, a Tor-powered botnet straight from Reddit. Retrieved from <https://community.rapid7.com/community/infosec/blog/2012/12/06/skynet-a-tor-powered-botnet-straight-from-reddit>

- Haddadi, F., & Phan, D. (2016). How to Choose from Different Botnet Detection Systems ?, (AnNet).
- Hints, A. (2002). Fingerprinting Websites Using Traffic Analysis. In P. Dingedine, R, Syverson (Ed.) (pp. 0–7). Springer-Verlag, LNCS 2482.  
<https://doi.org/10.1007/3-540-36467-6>
- IETF®. (1996). SOCKS Protocol Version 5. Retrieved from  
<https://www.ietf.org/rfc/rfc1928.txt>
- IETF®. (1999). Diffie-Hellman Key Agreement Method. Retrieved from  
<https://www.ietf.org/rfc/rfc2631.txt>
- IETF®. (2008). The Transport Layer Security (TLS) Protocol. Retrieved from  
<https://tools.ietf.org/html/rfc5246>
- Kale, T. G., & Wu, C. (2016). Improving the Tor Traffic Distribution with Circuit Switching Method Satoshi Ohzahata and Toshihiko Kato, *1*, 105–106.
- Koch, R., Golling, M., & Rodosek, G. D. (2016). How Anonymous Is the Tor Network? A Long-Term Black-Box Investigation.
- Kugisaki, Y. (2007). Bot Detection based on Traffic Analysis Graduate School of Information Science Faculty of Information Science Faculty of Information Science, 303–306. <https://doi.org/10.1109/IPC.2007.91>
- Munson, L. (2013). Tor usage doubles in August. New privacy-seeking users or botnet? Retrieved from <https://nakedsecurity.sophos.com/2013/08/29/tor-usage-doubles-in-august-new-privacy-seeking-users-or-botnet/>
- Rodríguez-Gómez, R. a., Maciá-Fernández, G., & García-Teodoro, P. (2013). Survey and taxonomy of botnet research through life-cycle. *ACM Computing Surveys*, *45*(4), 1–33. <https://doi.org/10.1145/2501654.2501659>
- Sanatinia, A., & Noubir, G. (2015). OnionBots: Subverting Privacy Infrastructure for Cyber Attacks. In *Proceedings of the International Conference on Dependable Systems and Networks* (Vol. 2015–Septe, pp. 69–80).  
<https://doi.org/10.1109/DSN.2015.40>
- Schales, D. L., Jang, J., Wang, T., Hu, X., Kirat, D., Wuest, B., & Stoecklin, M. P. (2016). Scalable analytics to detect DNS misuse for establishing stealthy communication channels. *IBM Journal of Research and Development*, *60*(4), 1–14. <https://doi.org/10.1147/JRD.2016.2557639>
- Silva, S. S. C., Silva, R. M. P., Pinto, R. C. G., & Salles, R. M. (2013). Botnets: A survey. *Computer Networks*, *57*(2), 378–403.  
<https://doi.org/10.1016/j.comnet.2012.07.021>
- Springall, A., De Vito, C., & Huang, S. H. S. (2015). Per connection server-side identification of connections via Tor. In *Proceedings - International Conference on Advanced Information Networking and Applications, AINA* (Vol. 2015–April, pp. 727–734). <https://doi.org/10.1109/AINA.2015.260>
- Syverson, P., & Boyce, G. (2016). Bake in .onion for Tear-Free and Stronger Website Authentication. *IEEE Security and Privacy*, *14*(2), 15–21.  
<https://doi.org/10.1109/MSP.2016.33>
- Tansettanakorn, C., Thongprasit, S., Thamkongka, S., & Visoottiviseth, V. (2016). ABIS : A Prototype of Android Botnet Identification System.  
<https://doi.org/10.1109/ICT-ISPC.2016.7519221>
- Thakar, B. (2016). Advance Persistent Threat : Botnet.
- Tsudik, G., Uzun, E., & Wood, C. A. (2016). AC 3 N: Anonymous Communication in Content-Centric Networking, 1–4.
- Wang, T.-S., Lin, C.-S., & Lin, H.-T. (2016). DGA Botnet Detection Utilizing Social Network Analysis. *2016 International Symposium on Computer, Consumer and*

- Control (IS3C)*, 333–336. <https://doi.org/10.1109/IS3C.2016.93>
- Wang, T., Hu, X., Jang, J., Ji, S., Stoecklin, M., & Taylor, T. (2016). BotMeter: Charting DGA-Botnet Landscapes in Large Networks. *2016 IEEE 36th International Conference on Distributed Computing Systems (ICDCS)*, 334–343. <https://doi.org/10.1109/ICDCS.2016.77>
- Xv, V., April, P., Fossi, M., & Johnson, E. (2010). Symantec enterprise security Symantec Global Internet Security Threat Report Trends for 2009, XV(April).
- Yklijnsma. (2013). Large botnet cause of recent Tor network overload. Retrieved from <https://blog.fox-it.com/2013/09/05/large-botnet-cause-of-recent-tor-network-overload/>



## APPENDIX 1 – SYSTEM SETUP INSTRUCTIONS

### 1. Packages that need to be installed.

- 1.1. System designed for Linux based operating systems. Ubuntu 16.04lts version is used.
- 1.2. Command line for updating repositories.  
“\$ sudo apt-get update”
- 1.3. Command line for installing required programs and libraries.
  - 1.3.1. Python: Required programming language.
  - 1.3.2. LXC: Linux container systems.
  - 1.3.3. Openssl: Open source security toolkit.
  - 1.3.4. Libssl-dev: Necessary library for secure operations
  - 1.3.5. Hexchat: Internet relay chat (IRC) client
  - 1.3.6. TOR: The Onion Router (TOR) packages  
“\$ sudo apt-get install tor lxc python libssl-dev openssl hexchat”
- 1.1. Other open source required projects. They can be found in folder (Master-Project)
  - 1.1.1. Chutney: Open source Tor network simulation program for testing and configuring different kind of Tor networks. Chutney is changed from the original version.
  - 1.1.2. Unrealircd-4.0.9: IRC server with specialized configuration.

### 2. Implementation steps.

- 2.1. Setup Ubuntu. (standard Ubuntu implementation).
- 2.2. Implement specified code in “1.2“to command line.
- 2.3. Implement specified code in “1.3“to command line.
- 2.4. Configuring IRC server:
  - 2.4.1. Execute commands in “unrealircd-4.0.9/README” folder
  - 2.4.2. Copy “unrealircd.conf” to “./unrealircd/conf” . Necessary folder can be found in (Master-Project)
  - 2.4.3. Start IRC server “./unrealircd start”.
- 2.5. Configuring Tor network:
  - 2.5.1. Enter the folder “chutney” start customized Tor network configuration with “./chutney configure networks/hs-new”.
  - 2.5.2. Start network “./chutney start networks/hs-new”.

- 2.5.3. Check status `./chutney status networks/hs-new`. There must be seen 28 machine online.
  - 2.5.4. Then add the specified line `HiddenServicePort 6667 127.0.0.1:6667` to `./chutney/net/nodes/027h/torrc` end of the file. This line specifies hidden service port as a IRC server.
  - 2.5.5. Open `./chutney/net/nodes/027h/hidden_service/hostname` this address defines necessary web server's hostname.
- 2.6. Configure LXC's:
- 2.6.1. Container defined this section is there and compressed as `container_fs.tar.gz` decompress and cloning operations enough for implementation. 2.8.6.
  - 2.6.2. Create bot container with `sudo lxc-create -t Ubuntu -n bot0` command.
  - 2.6.3. Start container with `sudo lxc-start -n bot0` command
  - 2.6.4. Copy `/Master-Project/bot_code/bot.py` to `/var/lib/lxc/bot0/rootfs/` necessary code for bots.
  - 2.6.5. Change `.online` link in `/bot.py` line 54. With url in 2.5.5.
  - 2.6.6. Copy `/Master-Project/bot_code/start.sh` to `/var/lib/lxc/bot0/rootfs/` necessary code for automatized work.
  - 2.6.7. Add `/start.sh` lines to `/var7lib/lxc/bot0/rootfs/etc/rc.local`
  - 2.6.8. Bot0 is ready, clone bot0 for increasing number of bots. `sudo lxc-clone -o bot0 -n bot1` repeat n times for preferred botnet size.
- 2.7. Configure Hexchat:
- 2.7.1. Go `Preferences>Network setup>Proxy Server` and change `Hostname` to `127.0.0.1` , `Port` to `9050` and `Type` to `Socks` press `OK` and close the window.
  - 2.7.2. Go `Network List>add` and edit url at top of the window to specified Hidden service address defined in section 2.5.5.
  - 2.7.3. Wait ~5-10 min to network creation and bots.
- 2.8. Usage: 2 simple command can be used in this version.
- 2.8.1. `discover`
  - 2.8.2. `attack`

## APPENDIX 2 – SAMPLE CODES

### Sample Codes of Bot

```
from twisted.internet import reactor
from twisted.internet.protocol import Factory, Protocol
from twisted.internet.endpoints import TCP4ClientEndpoint
from twisted.internet import protocol
from twisted.words.protocols import irc
from txsocksx.client import SOCKS5ClientEndpoint
import random

class MomBot(irc.IRCClient):
    def _get_nickname(self):
        return self.factory.nickname
    nickname = property(_get_nickname)

    def signedOn(self):
        self.join(self.factory.channel)
        f = open('/a.log', 'w')
        f.write("Signed on as %s." % (self.nickname,))
        print "Signed on as %s." % (self.nickname,)
        f.closed
```

```

def joined(self, channel):
    print "Joined %s." % (channel,)

def privmsg(self, user, channel, msg):
    print msg
    if msg == "discover":
        self.msg(channel, "nmap sample")
    elif msg == "attack":
        self.msg(channel, "attack sample ")

class MomBotFactory(protocol.ClientFactory):
    protocol = MomBot

    def __init__(self, channel, nickname):
        self.channel = channel
        self.nickname = nickname

    def clientConnectionLost(self, connector, reason):
        print "Lost connection (%s), reconnecting." % (reason,)
        connector.connect()

    def clientConnectionFailed(self, connector, reason):
        print "Could not connect: %s" % (reason,)

```

## Botnet Trigger Code

```
#!/bin/bash

pkill tor
tor &
rm /a.log
touch /a.log
file="/a.log"
sleep 20

while true; do
    echo "try to connecting server"
    if [ -s "$file" ]; then
        echo "done. "
        exit 0
    else
        pkill python
        echo "try moar."
        python /a.py & sleep 30
    fi
done
```

## APPENDIX 3 – TOR CONFIGURATION SAMPLES

### Authority Server Tor Config File (There are 4 more)

```
TestingTorNetwork 1

## Comprehensive Bootstrap Testing Options ##
# These typically launch a working minimal Tor network in 25s-30s,
# and a working HS Tor network in 40-45s.
# See authority.tmpl for a partial explanation
#AssumeReachable 0
#Default PathsNeededToBuildCircuits 0.6
#Disable TestingDirAuthVoteExit
#Disable TestingDirAuthVoteHSDir
#Default V3AuthNIntervalsValid 3

## Rapid Bootstrap Testing Options ##
# These typically launch a working minimal Tor network in 6s-10s
# These parameters make tor networks bootstrap fast,
# but can cause consensus instability and network unreliability
# (Some are also bad for security.)
AssumeReachable 1
PathsNeededToBuildCircuits 0.25
TestingDirAuthVoteExit *
TestingDirAuthVoteHSDir *
V3AuthNIntervalsValid 2
```

```
## Always On Testing Options ##
# We enable TestingDirAuthVoteGuard to avoid Guard stability requirements
TestingDirAuthVoteGuard *
# We set TestingMinExitFlagThreshold to 0 to avoid Exit bandwidth requirements
TestingMinExitFlagThreshold 0
# VoteOnHidServDirectoriesV2 needs to be set for HSDirs to get the HSDir flag
#Default VoteOnHidServDirectoriesV2 1

## Options that we always want to test ##
Sandbox 1

DataDirectory /home/hus/Desktop/chutney/net/nodes/000a
RunAsDaemon 1
ConnLimit 60
Nickname test000a
ShutdownWaitLength 0
PidFile /home/hus/Desktop/chutney/net/nodes/000a/pid
Log notice file /home/hus/Desktop/chutney/net/nodes/000a/notice.log
Log info file /home/hus/Desktop/chutney/net/nodes/000a/info.log
# Turn this off to save space
#Log debug file /home/hus/Desktop/chutney/net/nodes/000a/debug.log
ProtocolWarnings 1
SafeLogging 0
DisableDebuggerAttachment 0
DirAuthority test000a orport=5000 no-v2 hs
v3ident=3028F321F8F02FC07EE85667F4726E8FCCF37ACC 192.168.1.100:7000
F23ADC68E07AD5B54FB6908A067E3ACDE967E041
DirAuthority test001a orport=5001 no-v2 hs
v3ident=A21F7263060BDCB770C4662C63025007D8414DD6 192.168.1.100:7001
879C65A8207700D06F551B5FE5DD3AD31305DEE2
```

```
DirAuthority test002a orport=5002 no-v2 hs
v3ident=3258059F1B891488BED6368D090027A4440F52D4 192.168.1.100:7002
7B74CAC4FC6D8361E1F76F1B6EFC07DB48ECF8CF

DirAuthority test003a orport=5003 no-v2 hs
v3ident=BF80E4839FE954C41D631AED7CB228C7DDF4028C 192.168.1.100:7003
5019A6872671BF5F8360B6C9DE75673B4D5B5E1C

ControlPort 8000
SocksPort 0
OrPort 5000
Address 192.168.1.100
DirPort 7000

#NOTE: Setting TestingServerConsensusDownloadSchedule doesn't
# help -- dl_stats.schedule is not DL_SCHEDULE_CONSENSUS
# at bootstrap time.
# Try to download after:
# the minimum initial consensus time to start with,
# a few eager fetches,
# then half the minimum testing consensus interval
TestingServerDownloadSchedule 0, 5
TestingServerConsensusDownloadSchedule 0, 5

AuthoritativeDirectory 1
V3AuthoritativeDirectory 1
ContactInfo auth0@test.test

# Speed up the consensus cycle as fast as it will go
# Voting Interval can be:
# 10, 12, 15, 18, 20, 24, 25, 30, 36, 40, 45, 50, 60, ...
# Testing Initial Voting Interval can be:
# 5, 6, 8, 9, or any of the possible values for Voting Interval,
# as they both need to evenly divide 30 minutes.
```



```
# If clock desynchronisation is an issue, use an interval of at least:
# 18 * drift in seconds, to allow for a clock slop factor
TestingV3AuthInitialVotingInterval 5
V3AuthVotingInterval 10
# VoteDelay + DistDelay must be less than VotingInterval
TestingV3AuthInitialVoteDelay 2
V3AuthVoteDelay 2
TestingV3AuthInitialDistDelay 2
V3AuthDistDelay 2
# This is autoconfigured by chutney, so you probably don't want to use it
#TestingV3AuthVotingStartOffset 0

# Work around situations where the Exit, Guard and HSDir flags aren't being set
# These flags are all set eventually, but it takes Guard up to ~30 minutes
# We could be more precise here, but it's easiest just to vote everything
# Clients are sensible enough to filter out Exits without any exit ports,
# and Guards and HSDirs without ORPorts
# If your tor doesn't recognise TestingDirAuthVoteExit/HSDir,
# either update your chutney to a 2015 version,
# or update your tor to a later version, most likely 0.2.6.2-final

# These are all set in common.i in the Comprehensive/Rapid sections
# Work around Exit requirements
#TestingDirAuthVoteExit *
# Work around bandwidth thresholds for exits
#TestingMinExitFlagThreshold 0
# Work around Guard uptime requirements
#TestingDirAuthVoteGuard *
# Work around HSDir uptime and ORPort connectivity requirements
#TestingDirAuthVoteHSDir *
```

## Relay Tor Config File (There are 20 more)

```
TestingTorNetwork 1

## Comprehensive Bootstrap Testing Options ##
# These typically launch a working minimal Tor network in 25s-30s,
# and a working HS Tor network in 40-45s.
# See authority.tmpl for a partial explanation
#AssumeReachable 0
#Default PathsNeededToBuildCircuits 0.6
#Disable TestingDirAuthVoteExit
#Disable TestingDirAuthVoteHSDir
#Default V3AuthNIntervalsValid 3

## Rapid Bootstrap Testing Options ##
# These typically launch a working minimal Tor network in 6s-10s
# These parameters make tor networks bootstrap fast,
# but can cause consensus instability and network unreliability
# (Some are also bad for security.)
AssumeReachable 1
PathsNeededToBuildCircuits 0.25
TestingDirAuthVoteExit *
TestingDirAuthVoteHSDir *
V3AuthNIntervalsValid 2
```

```
## Always On Testing Options ##
# We enable TestingDirAuthVoteGuard to avoid Guard stability requirements
TestingDirAuthVoteGuard *
# We set TestingMinExitFlagThreshold to 0 to avoid Exit bandwidth requirements
TestingMinExitFlagThreshold 0
# VoteOnHidServDirectoriesV2 needs to be set for HSDirs to get the HSDir flag
#Default VoteOnHidServDirectoriesV2 1
## Options that we always want to test ##
Sandbox 1

DataDirectory /home/hus/Desktop/chutney/net/nodes/004r
RunAsDaemon 1
ConnLimit 60
Nickname test004r
ShutdownWaitLength 0
PidFile /home/hus/Desktop/chutney/net/nodes/004r/pid
Log notice file /home/hus/Desktop/chutney/net/nodes/004r/notice.log
Log info file /home/hus/Desktop/chutney/net/nodes/004r/info.log
# Turn this off to save space
#Log debug file /home/hus/Desktop/chutney/net/nodes/004r/debug.log
ProtocolWarnings 1
SafeLogging 0
DisableDebuggerAttachment 0
DirAuthority test000a orport=5000 no-v2 hs
v3ident=3028F321F8F02FC07EE85667F4726E8FCCF37ACC 192.168.1.100:7000
F23ADC68E07AD5B54FB6908A067E3ACDE967E041
DirAuthority test001a orport=5001 no-v2 hs
v3ident=A21F7263060BDCB770C4662C63025007D8414DD6 192.168.1.100:7001
879C65A8207700D06F551B5FE5DD3AD31305DEE2
DirAuthority test002a orport=5002 no-v2 hs
v3ident=3258059F1B891488BED6368D090027A4440F52D4 192.168.1.100:7002
7B74CAC4FC6D8361E1F76F1B6EFC07DB48ECF8CF
```

```
DirAuthority test003a orport=5003 no-v2 hs
v3ident=BF80E4839FE954C41D631AED7CB228C7DDF4028C 192.168.1.100:7003
5019A6872671BF5F8360B6C9DE75673B4D5B5E1C

ControlPort 8004

SocksPort 0

OrPort 5004

Address 192.168.1.100

DirPort 7004

#NOTE: Setting TestingServerConsensusDownloadSchedule doesn't
#   help -- dl_stats.schedule is not DL_SCHED_CONSENSUS
#   at bootstrap time.
# Try to download after:
# the minimum initial consensus time to start with,
# a few eager fetches,
# then half the minimum testing consensus interval

TestingServerDownloadSchedule 0, 5

TestingServerConsensusDownloadSchedule 0, 5
```

## Client Tor Config File (there are 2 more)

```
TestingTorNetwork 1

## Comprehensive Bootstrap Testing Options ##
# These typically launch a working minimal Tor network in 25s-30s,
# and a working HS Tor network in 40-45s.
# See authority.tmpl for a partial explanation
#AssumeReachable 0
#Default PathsNeededToBuildCircuits 0.6
#Disable TestingDirAuthVoteExit
#Disable TestingDirAuthVoteHSDir
#Default V3AuthNIntervalsValid 3

## Rapid Bootstrap Testing Options ##
# These typically launch a working minimal Tor network in 6s-10s
# These parameters make tor networks bootstrap fast,
# but can cause consensus instability and network unreliability
# (Some are also bad for security.)
AssumeReachable 1
PathsNeededToBuildCircuits 0.25
TestingDirAuthVoteExit *
TestingDirAuthVoteHSDir *
V3AuthNIntervalsValid 2
```

```

## Always On Testing Options ##
# We enable TestingDirAuthVoteGuard to avoid Guard stability requirements
TestingDirAuthVoteGuard *
# We set TestingMinExitFlagThreshold to 0 to avoid Exit bandwidth requirements
TestingMinExitFlagThreshold 0
# VoteOnHidServDirectoriesV2 needs to be set for HSDirs to get the HSDir flag
#Default VoteOnHidServDirectoriesV2 1

## Options that we always want to test ##
Sandbox 1

DataDirectory /home/hus/Desktop/chutney/net/nodes/024c
RunAsDaemon 1
ConnLimit 60
Nickname test024c
ShutdownWaitLength 0
PidFile /home/hus/Desktop/chutney/net/nodes/024c/pid
Log notice file /home/hus/Desktop/chutney/net/nodes/024c/notice.log
Log info file /home/hus/Desktop/chutney/net/nodes/024c/info.log
# Turn this off to save space
#Log debug file /home/hus/Desktop/chutney/net/nodes/024c/debug.log
ProtocolWarnings 1
SafeLogging 0
DisableDebuggerAttachment 0

DirAuthority test000a orport=5000 no-v2 hs
v3ident=3028F321F8F02FC07EE85667F4726E8FCCF37ACC 192.168.1.100:7000
F23ADC68E07AD5B54FB6908A067E3ACDE967E041

DirAuthority test001a orport=5001 no-v2 hs
v3ident=A21F7263060BDCB770C4662C63025007D8414DD6 192.168.1.100:7001
879C65A8207700D06F551B5FE5DD3AD31305DEE2

DirAuthority test002a orport=5002 no-v2 hs
v3ident=3258059F1B891488BED6368D090027A4440F52D4 192.168.1.100:7002
7B74CAC4FC6D8361E1F76F1B6EFC07DB48ECF8CF

```

```
DirAuthority test003a orport=5003 no-v2 hs  
v3ident=BF80E4839FE954C41D631AED7CB228C7DDF4028C 192.168.1.100:7003  
5019A6872671BF5F8360B6C9DE75673B4D5B5E1C
```

```
ControlPort 8024
```

```
SocksPort 9024
```

```
CookieAuthentication 1
```

```
#NOTE: Setting TestingClientConsensusDownloadSchedule doesn't
```

```
# help -- dl_stats.schedule is not DL_SCHED_CONSENSUS
```

```
# at bootstrap time.
```

```
# Try to download after:
```

```
# the minimum initial consensus time to start with,
```

```
# a few eager fetches,
```

```
# then half the minimum testing consensus interval
```

```
TestingClientDownloadSchedule 0, 5
```

```
TestingClientConsensusDownloadSchedule 0, 5
```

## Hidden Service Tor Config File

```
TestingTorNetwork 1

## Comprehensive Bootstrap Testing Options ##
# These typically launch a working minimal Tor network in 25s-30s,
# and a working HS Tor network in 40-45s.
# See authority.tmpl for a partial explanation
#AssumeReachable 0
#Default PathsNeededToBuildCircuits 0.6
#Disable TestingDirAuthVoteExit
#Disable TestingDirAuthVoteHSDir
#Default V3AuthNIntervalsValid 3

## Rapid Bootstrap Testing Options ##
# These typically launch a working minimal Tor network in 6s-10s
# These parameters make tor networks bootstrap fast,
# but can cause consensus instability and network unreliability
# (Some are also bad for security.)
AssumeReachable 1
PathsNeededToBuildCircuits 0.25
TestingDirAuthVoteExit *
TestingDirAuthVoteHSDir *
V3AuthNIntervalsValid 2
```



```

## Always On Testing Options ##
# We enable TestingDirAuthVoteGuard to avoid Guard stability requirements
TestingDirAuthVoteGuard *
# We set TestingMinExitFlagThreshold to 0 to avoid Exit bandwidth requirements
TestingMinExitFlagThreshold 0
# VoteOnHidServDirectoriesV2 needs to be set for HSDirs to get the HSDir flag
#Default VoteOnHidServDirectoriesV2 1

## Options that we always want to test ##
Sandbox 1

DataDirectory /home/hus/Desktop/chutney/net/nodes/027h
RunAsDaemon 1
ConnLimit 60
Nickname test027h
ShutdownWaitLength 0
PidFile /home/hus/Desktop/chutney/net/nodes/027h/pid
Log notice file /home/hus/Desktop/chutney/net/nodes/027h/notice.log
Log info file /home/hus/Desktop/chutney/net/nodes/027h/info.log
# Turn this off to save space
#Log debug file /home/hus/Desktop/chutney/net/nodes/027h/debug.log
ProtocolWarnings 1
SafeLogging 0
DisableDebuggerAttachment 0

DirAuthority test000a orport=5000 no-v2 hs
v3ident=3028F321F8F02FC07EE85667F4726E8FCCF37ACC 192.168.1.100:7000
F23ADC68E07AD5B54FB6908A067E3ACDE967E041

DirAuthority test001a orport=5001 no-v2 hs
v3ident=A21F7263060BDCB770C4662C63025007D8414DD6 192.168.1.100:7001
879C65A8207700D06F551B5FE5DD3AD31305DEE2

DirAuthority test002a orport=5002 no-v2 hs
v3ident=3258059F1B891488BED6368D090027A4440F52D4 192.168.1.100:7002
7B74CAC4FC6D8361E1F76F1B6EFC07DB48ECF8CF

```

```
DirAuthority test003a orport=5003 no-v2 hs  
v3ident=BF80E4839FE954C41D631AED7CB228C7DDF4028C 192.168.1.100:7003  
5019A6872671BF5F8360B6C9DE75673B4D5B5E1C
```

```
ControlPort 8027
```

```
SocksPort 0
```

```
Address 192.168.1.100
```

```
HiddenServiceDir /home/hus/Desktop/chutney/net/nodes/027h/hidden_service
```

```
# SSH is usually a popular service that is running. This is really just to make
```

```
# a quick way to test the HS with torsocks and have an app at the other end.
```

```
#HiddenServicePort 6000 127.0.0.1:22
```

```
# Redirect requests to the port used by chutney verify
```

```
HiddenServicePort 6667 127.0.0.1:6667
```

```
#HiddenServicePort 5858 127.0.0.1:4747
```

## APPENDIX 4 – OPEN SOURCE PROGRAMS

### Chutney

Open source tool for configure and testing Tor network.

Link: <https://github.com/torproject/chutney>

### LXC

LXC is an operating-system-level virtualization method for running multiple isolated Linux systems on a control host.

Link: <https://linuxcontainers.org>

### HexChat

Chat program based on an IRC protocol.

Link: <https://hexchat.github.io>