

12-2017

Multi-Stage Detection Technique for DNS-Based Botnets

Wasseem Jammal

St. Cloud State University, wmjammal@stcloudstate.edu

Follow this and additional works at: https://repository.stcloudstate.edu/msia_etds

Recommended Citation

Jammal, Wasseem, "Multi-Stage Detection Technique for DNS-Based Botnets" (2017). *Culminating Projects in Information Assurance*. 38.

https://repository.stcloudstate.edu/msia_etds/38

This Thesis is brought to you for free and open access by the Department of Information Systems at theRepository at St. Cloud State. It has been accepted for inclusion in Culminating Projects in Information Assurance by an authorized administrator of theRepository at St. Cloud State. For more information, please contact rswexelbaum@stcloudstate.edu.

Multi-Stage Detection Technique for DNS-Based Botnets

by

Wasseem Jammal

A Thesis

Submitted to the Graduate Faculty of

St. Cloud State University

in Partial Fulfillment of the Requirements

for the Degree of

Master of Science

in Information Assurance

December, 2017

Thesis Committee:
Tirthankar Ghosh, Chairperson
Mark Schmidt
Mehdi Mekni

Abstract

Domain Name System (DNS) is one of the most widely used protocols in the Internet. The main purpose of the DNS protocol is mapping user-friendly domain names to IP addresses. Unfortunately, many cyber criminals deploy the DNS protocol for malicious purposes, such as botnet communications. In this type of attack, the botmasters tunnel communications between the Command and Control (C&C) servers and the bot-infected machines within DNS request and response. Designing an effective approach for botnet detection has been done previously based on specific botnet types. Since botnet communications are characterized by different features, botmasters may evade detection methods by modifying some of these features. This research aims to design and implement a multi-staged detection approach for Domain Generation Algorithm (DGA), Fast Flux Service Network, and Domain Flux-based botnets, as well as encrypted DNS tunneled-based botnets using the BRO Network Security Monitor. This approach is able to detect DNS-based botnet communications by relying on analyzing different techniques used for finding the C&C server, as well as encrypting the malicious traffic.

Acknowledgement

First and foremost, I would like to express my sincere gratitude to my advisor, Dr. Tirthankar Ghosh, for his support of my study and research. He has been supportive since the days I began taking his classes on the Intrusion Detection & Prevention Systems, Firewalls, and Penetration Testing. In his lectures and projects, I remember he used to say, "In security, you have to think like a bad guy and act like a good guy " to promote our critical thinking as graduate students. Besides my advisor, I would like to thank the rest of my thesis committee, Dr. Mark Schmidt and Dr. Mehdi Mekni, for their encouragement and insightful comments.

I would also like to express my profound thanks to my parents, brothers, and sisters for providing me with unfailing support and continuous encouragement throughout my life.

Last, but not the least, I would like to thank my wife, Amal, for her sincere support. I would never have been able to finish my thesis without her love, support, and patience.

Thank you very much, everyone!

Table of Contents

	Page
List of Tables	6
List of Figures	7
Chapter	
I. Introduction	8
Introduction	8
Problem Statement	10
Nature and Significance of the Problem	11
Objective of the Study	12
Limitations of the Study	13
Definition of Terms	13
Summary	15
II. Background and Review of Literature	16
Introduction	16
Background Related to the Problem	16
Literature Review	22
Summary	32
III. Methodology	33
Introduction	33
Design of the Study	33
Data Collection.....	37

Chapter	Page
Data Analysis	38
Summary	38
IV. Data Presentation and Analysis	39
Introduction	39
Data Presentation	39
Data Analysis	41
Summary	49
V. Results, Conclusion, and Recommendations	50
Introduction	50
Results	50
Conclusion	53
Future Work	53
References	55
Appendices:	
A. DNA Message Format	60
B. Iodine Lab Implementation	68
C. DNS2TCP Lab Implementation	73
D. BRO Network Security Monitor Scripts	77
E. DGA-, FFSN-, and DF-based Botnets Dataset	82

List of Tables

Table	Page
1. Variety of Botnet Automated Activities	12
2. Definition of Terms Used in This Document	13
3. Domain Flux Implementation	20
4. FFSN & DF Implementation	21
5. Feature Vector	23
6. Differences between Botnet and Legitimate DNS	24

List of Figures

Figure	Page
1. Botnet architectures	18
2. Fast-Flux service network	20
3. DNS tunneling	22
4. Functioning principle of the DNS-based anti-evasion technique for botnet detection	26
5. A High-Level Overview of Pleiades	29
6. DNS statistics	35
7. Multistage detection technique for DNS-based botnets	35
8. Iodine and DNS2TCP lab	41
9. DNS packets statistics–DGA-based botnet	42
10. Wireshark packet capture of DGA-based botnet	43
11. DNS packets statistics–FFSN & DF-based botnet	43
12. Domain flux implementation	44
13. FFSN implementation	45
14. Iodine connection negotiation	45
15. SSH connection tunneled in DNS packets (iodine)	47
16. SSH connection tunneled in DNS packets (dns2tcp)	48

Chapter I: Introduction

Introduction

Cyber-attacks, including malware injection, never stop threatening computer networks and information systems. One of the predominant forms of spreading malware is infecting systems with malicious software forcing them to act as botnets. These botnets often utilize Domain Name System (DNS) protocol to hide their communication with their Command & Control (C&C) servers. This research aims to design and implement a multi-staged detection approach for Domain Generation Algorithm (DGA), Fast Flux Service Network, and Domain Flux-based botnets, as well as encrypted DNS tunneled-based botnets using BRO Network Security Monitor.

Domain Name System is the Internet's equivalent of a phone book and a central part of the Internet, providing a mechanism for naming resources in such a way that the names are usable in different hosts, networks, protocol families, internets, and administrative organizations (RFC 883). In other words, DNS translates more readily memorized domain names to numerical IP addresses needed for locating and identifying computing devices and services (Domain Name System, 2017).

According to Wikipedia, mapping a simpler and more memorable name to a numerical address dates back to the ARPANET era (1969-1990). The Stanford Research Institute (SRI) maintained a text file named HOSTS.TXT to map host names to the numerical addresses of computers on the ARPANET. Each time a host had to connect to the network, it would have to download the latest version of the

table. At that time, the addresses were assigned manually, but by the early 1980s, keeping a single and centralized host table had turned out to become slow and unwieldy, creating the need for an automated naming system. The Domain Name System was created by Paul Mockapetris and the Internet Engineering Task Force (IETF) published the original specifications in RFC 882 and RFC 883 in November 1983 (Domain Name System, 2017).

A botnet, or robot network, is a collection of Internet-connected computers (bots) that are infected with a specific malware that allows these bots to be remotely controlled by a Command and Control (C&C) server (What is a Botnet Attack?- Definition, n.d.). Although DNS protocol is typically used for benign purposes, it can be used to port malicious packets. Domain Name System protocol can be abused at different stages of the communication process with botnets, which may implement some techniques to circumvent the detection methods.

Domain Generation Algorithm (DGA), Fast-Flux Service Network (FFSN), Domain Flux¹, Double Flux, and DNS tunneling can be used as evasion techniques in botnet communications (Dietrich et al., 2011; Farnham & Atlasis, 2013; Lysenko, Pomorova, Savenko, Kryshchuk, & Bobrovnikova, 2015). In DGA, a large number of short-lived domain names are generated. While the DNS A-record (IP address) for a specific malicious domain name changes frequently in FFSN, the DF changes the C&C server's domain name repeatedly by implementing short TTL period. With DNS

¹ DF will be used in this document to refer to Domain Flux.

tunneling, botnet communications with C&C servers can be wrapped and tunneled through DNS packets.

BRO Network Security Monitor is a powerful and open-source network analysis framework (The Bro Network Security Monitor, n.d.). The initial version of BRO was designed and implemented by Vern Paxson in 1995. BRO focuses on network security monitoring, and provides a comprehensive platform for more general network traffic analysis tasks even outside of the security domain, including performance measurements and helping with troubleshooting.

BRO is not a classic signature-based intrusion detection system (IDS). While it supports such standard functionality as well, BRO supports a wide range of analyses through its scripting language which indeed facilitates a much broader spectrum of very different approaches to finding malicious activity, including semantic misuse detection, anomaly detection, and behavioral analysis (Bro Introduction, n.d.).

Problem Statement

Different botnets implement different protocols to communicate with C&C servers. As the Domain Name System (DNS) is one of the least monitored protocols from a security perspective, many cyber criminals abuse DNS to tunnel botnet communications, and some of the tunneling tools may encrypt the payload to evade detection. According to Dietrich et al. (2011), "DNS is usually one of the few protocols- if not the only one- that is allowed to pass without further ado" (p. 10), thus making it an attractive means of botnet tunneling. Furthermore, many botnet detection methods can be evaded by some techniques such as DGA, FFSN, and DF

(Lysenko et al., 2015). This makes the detection and prevention efforts more complicated.

Domain Name System protocol can be abused at different stages of the botnet communication process, such as finding the C&C server, transmitting data, and/or controlling the bots. At the initial stage of botnet communication, a bot tries to find its C&C server by sending DNS requests to resolve the domain name(s) of the malicious server. After finding the C&C server, the communications between C&C server and the bot start. These communications may include data exfiltration, data infiltration, or controlling the bot to perform malicious actions against other systems.

Nature and Significance of the Problem

Botnets have become one of the most significant concerns on the Internet. "According to a report from Russian-based Kaspersky Labs, botnets—not spam, viruses, or worms ... pose the biggest threat to the Internet. A report from Symantec came to a similar conclusion" (Newman, 2010). Botnet malware can be used against any Internet connected device, including smart televisions, to execute a wide range of malicious actions. These actions include launching a distributed denial-of-service (DDoS), phishing, generating and sending spam messages, propagating malware, sniffing information, hosting malicious content, and using infected bots for proxy activities (Bots and Botnets—A Growing Threat, 2016). Table 1 shows some malicious activities that are executed against or by botnet-infected machines.

Table 1

Variety of Botnet Automated Activities

Data exfiltration against bots	Data infiltration against bots	Activities performed by bots
Stealing sensitive information	Installing malware Sending spams	Lunching DoS Click fraud Proxy activities

The resiliency and dynamic nature of botnets pose challenges to detection methods, like when the botmasters change the botnets' characteristics to avoid detection methods. Designing and implementing a detection mechanism for DNS-based botnets will rely on analysis of botnet features, and implementation of detection methods and existing evasion techniques.

Objective of the Study

Using BRO Network Security Monitor, the main objective of this study is to design and implement an effective detection technique for DNS-based botnets using rule-based and signature-based techniques in two phases. The first phase will implement a rule-based mechanism to detect the existence of DGA, FFSN, or DF based botnets, which are utilized to find the C&C server. The second phase will implement a signature-based mechanism to detect DNS tunneling, which is utilized to wrap malicious traffic in DNS packets.

Limitations of the Study

This study required a group of real botnet-infected machines and C&C servers. Network traffic of previously infected machines were used for analysis and design of the proposed detection mechanism of the DGA, FFSN, and DF implementations.

Definition of Terms

Table 2

Definition of Terms Used in This Document

Term	Definition
Authoritative Name Server (NS)	<p>A name server that provides actual, original and definitive answer to DNS queries such as – mail server IP address (MX resource record) or web site IP address (A resource record). It does not provide just cached answers that were obtained from another name server.</p> <p>What Is Authoritative Name Server? (2009, August 17). Retrieved March 25, 2017, from https://www.dnsknowledge.com/whatis/authoritative-nameserver/</p>
Botmaster	<p>A person who operates the command and control of botnets for remote process execution.</p> <p>DDoS Attack Definitions - DDoSPedia. (n.d.). Retrieved March 25, 2017, from https://security.radware.com/ddos-knowledgecenter/ddospedia/botmaster/</p>
Botnet (Zombie Army)	<p>An interconnected network of computers infected with malware without the user's knowledge and controlled by cybercriminals.</p> <p>What is a Botnet Attack? - Definition. (n.d.). Retrieved March 25, 2017, from https://usa.kaspersky.com/internet-securitycenter/threats/botnet-attacks#.WN3sKYWcE2w</p>
Command and Control Server (C&C)	<p>A computer that controls and issues commands to members of a botnet. Botnet members may be referred to zombies and the botnet itself may be referred to as a zombie army.</p> <p>What is command-and-control servers (C&C center)? - Definition from WhatIs.com. (n.d.). Retrieved March 25, 2017, from http://whatis.techtarget.com/definition/command-andcontrol-server-CC-server</p>
Covert Channel	<p>A mechanism for sending and receiving information data between machines without alerting any firewalls and IDS's on the network.</p> <p>IDFAQ: What is covert channel and what are some examples?</p> <p>(n.d.). Retrieved March 25, 2017, from https://www.sans.org/security-resources/idfaq/what-is-covertchannel-and-what-are-some-examples/2/17</p>
Distributed Denial of Service	<p>A type of DoS attack where multiple compromised systems, which are often infected with a Trojan, are used to target a single system causing a</p>

	<p>Denial of Service (DoS) attack. Victims of a DDoS attack consist of both the end targeted system and all systems maliciously used and controlled by the hacker in the distributed attack.</p> <p>Beal, V. (n.d.). DDoS attack - Distributed Denial of Service. Retrieved March 25, 2017, from http://www.webopedia.com/TERM/D/DDoS_attack.html</p>
DNS Tunneling	<p>The ability to encode the data of other programs or protocols in DNS queries and responses.</p> <p>What is DNS Tunneling? (2017, January 04). Retrieved March 25, 2017, from https://www.plixer.com/blog/networksecurity-forensics/what-is-dns-tunneling/</p>
Domain Flux (DF)	<p>A technique for keeping a malicious botnet in operation by constantly changing the domain name of the botnet owner's Command and Control (C&C) server.</p> <p>What is domain fluxing? - Definition from WhatIs.com. (n.d.). Retrieved March 25, 2017, from http://searchsecurity.techtarget.com/definition/domain-fluxing</p>
Domain Generation Algorithm (DGA)	<p>A class of algorithm that takes a seed as an input, outputs a string and appends a top-level domain (TLD) such as .com, .ru, .uk, etc. in order to form a possible domain name. The seed is a piece of information accessible to both the botmaster and the infected host now acting as a bot.</p> <p>Why Domain Generating Algorithms (DGAs)? -. (2016, August 17). Retrieved March 25, 2017, from http://blog.trendmicro.com/domain-generating-algorithms-dgas/</p>
Double Flux	<p>A DNS technique used by botnets to provide an additional layer of redundancy by changing the DNS A-records and authoritative NS-records continually for malicious domain using the round robin algorithm.</p> <p>Fast Flux Networks Working and Detection, Part 1. (2015, February 13).</p> <p>Retrieved March 25, 2017, from http://resources.infosecinstitute.com/fast-flux-networks-workingdetection-part-1/#gref</p>
Dynamic Domain Name System (DDNS)	<p>A method of automatically updating a name server in the Domain Name System (DNS), often in real time, with the active DDNS configuration of its configured hostnames, addresses or other information.</p> <p>Dynamic DNS. (2017, February 18). Retrieved March 25, 2017, from https://en.wikipedia.org/wiki/Dynamic_DNS</p>
Fast Flux or Single Flux (FFSN)	<p>A DNS technique used by botnets to associate a single domain name with many IP addresses and to hide phishing and malware delivery sites behind an ever-changing network of compromised hosts acting as proxies.</p> <p>Fast flux. (2017, March 20). Retrieved March 25, 2017, from https://en.wikipedia.org/wiki/Fast_flux</p>
Metamorphic and polymorphic malware	<p>Two categories of malicious software programs that have the ability to change their code as they propagate. With polymorphism, each time the bot binary propagates, it encrypts its original code to avoid pattern recognition. Instead of the code encryption, metamorphism changes the code to an equivalent one each time.</p> <p>What is metamorphic and polymorphic malware? - Definition from WhatIs.com. (n.d.). Retrieved March 26, 2017, from</p>

	http://searchsecurity.techtarget.com/definition/metamorphicand-polymorphic-malware
Network Address Translation (NAT)	<p>The process where a network device, usually a firewall, assigns a public address to a computer (or group of computers) inside a private network. The main use of NAT is to limit the number of public IP addresses an organization or company must use, for both economy and security purposes.</p> <p>What is Network Address Translation (NAT)? (n.d.). Retrieved March 25, 2017, from http://whatismyipaddress.com/nat</p>
Resource Records	<p>The data elements that define the structure and content of the domain name space. All DNS operations are ultimately formulated in terms of resource records.</p> <p>Resource Records. (n.d.). Retrieved March 25, 2017, from http://www.freesoft.org/CIE/Course/Section2/8.htm</p>
Single Flux	<p>The simplest type of fast flux, characterized by multiple individual nodes within the network registering and de-registering their IP addresses as part of the DNS A (address) record list for a single domain name. This combines round robin DNS with very short time to live - usually less than five minutes - to create a constantly changing list of destination addresses for that single DNS name. Fast flux. (2017, March 08). Retrieved March 25, 2017, from https://en.wikipedia.org/wiki/Fast_flux#Single-flux_and_doubleflux</p>
Zombie	<p>A computer connected to the Internet that has been compromised by a hacker, computer virus or Trojan horse program, and can be used to perform malicious tasks of one sort or another under remote direction. Zombie (computer science). (2017, March 22). Retrieved March 25, 2017, from https://en.wikipedia.org/wiki/Zombie_(computer_science)</p>

Summary

This chapter covered an introduction about botnets and their significance. Also, DNS protocol and how it is abused in favor of botnet communication, as well as some common evasion techniques used to circumvent botnet-detection methods. The next chapter provides an overview of botnets, more details about DNS-based botnet-detection methods and evasion techniques with a detailed review of existing literature.

Chapter II: Background and Review of Literature

Introduction

Botnets use various communication protocols to tunnel and hide themselves from detection. Also, they may use encryption techniques to encrypt the tunnel itself. These evasion techniques complicate the detection process.

Research communities have proposed many different approaches for botnet detection. Many of these approaches are based on a specific type of botnet. Studying and analyzing different botnet features, implemented detection approaches, and evasion techniques will be helpful in designing and implementing a new approach for detecting DNS-based botnets at different stages of communications.

Background Related to the Problem

Most computers that are co-opted to serve in botnet are often home-based and are inadequately protected by an effective firewall or other safeguard (Rouse, 2012). According to Trend Micro, the two pieces of malware that started the botnet usage were Sub7 and Pretty Park—a Trojan and a Worm, respectively (CounterMeasures—A Security Blog, 2010). These malwares introduced the concept of a victim machine connecting to an IRC channel to listen for malicious commands. These two pieces of malware first surfaced in 1999 and botnet innovation has been constant since then. Steadily, botnets migrated away from the original IRC Command & Control (C&C) channel to communicate over HTTP, ICMP, SSL, and DNS ports, often using custom protocols.

Botnet structure has evolved over time to evade detection and disruption. Bots are traditionally constructed as clients which communicate via central servers. These bots connect to one or more servers through one or more domains, allowing the botmaster to perform with total control from a remote location. The centralized C&C model introduces a single point of failure; if the C&C domain is identified and dismantled, the botmaster loses control over the entire botnet (Antonakakis et al., 2012). To solve the problem of security, researchers and authorities target botnet domains and C&C servers. Many recent botnets now rely on peer-to-peer networks to communicate. These P2P bot programs perform the same actions as the client-server model without the need for a central server to communicate (Botnet, 2017).

In a peer-to-peer model, incoming connections to computers—that are behind a Network Address Translation (NAT) gateway, firewall, or proxy server—cannot be established. This would prevent most bots being connected to by other bots. In a client-server model, this obviously is not a problem as the bots connect to the server, so a peer-to-peer network still requires servers in a way (Peer-to-Peer Botnets for Beginners, 2016).

Bots that are not behind a proxy / NAT / firewall can accept incoming connections and act as servers. These bots are usually referred to as nodes or peers, whereas the bots that do not accept incoming connections are usually referred to as workers. In a peer-to-peer model, the workers connect to one or more nodes to receive command(s). These nodes are technically servers, and the workers are

distributed between many nodes. This scenario allows the workers to shift to another node if one is dismantled.

Peer-to-Peer botnets present more challenges for detection authorities; it is impractical to take all the nodes down since the nodes are legitimate devices. They cannot simply be seized like a server would be (Peer-to-Peer Botnets for Beginners, 2016). Figure 1 shows the two communication models: client-server model between workers and C&C server and peer-to-peer model between nodes and workers.

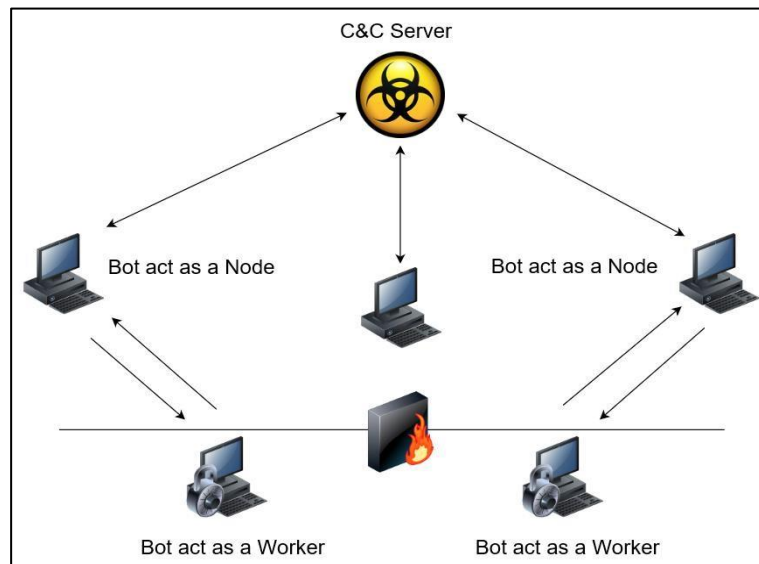


Figure 1. Botnet architectures.

Botnets core components include C&C server(s) and zombies. Upon successful infection, the infected machine tries to connect to the C&C server at the initial stage. There are different ways to find the C&C server: a) the IP address or the domain name of the C&C server is hard coded in the malware, b) the malware implements Domain Generation Algorithm (DGA), c) the malware implements Fast-Flux Service Network (FFSN), or d) the malware implements Domain Flux (DF). In

the hard coding implementation, it is easy for defenders, upon botnet discovery, to block a specific IP address or domain name. However, with DGA, it is difficult to block tens of thousands of unpredictable generated domain names, and the problem becomes more complicated when implementing DGA, FFSN, and DF together, thus, the traditional blacklisting technique based on the IP address or domain name is ineffective, and it is difficult to trace a large number of nodes ready to register their IP addresses to a domain name(s).

A botnet that implements DGA generates tens of thousands of domain names per day. These domains are short-lived and blacklists will not be effective. As generated domains are predictable to the botmaster, they need to register only one of the domains to initiate C&C connection, whereas defenders need to block any generated domains that are registered to completely eliminate C&C activity (Hagen & Luo, 2016).

In FFSN, the basic concept is having multiple IP addresses associated with a single domain name, and then constantly changing them in quick succession. If one or more of them drop, others quickly take their place (Albors, 2017). Figure 2 shows how botmasters use bots (flux agents) to act as proxies to the C&C server.

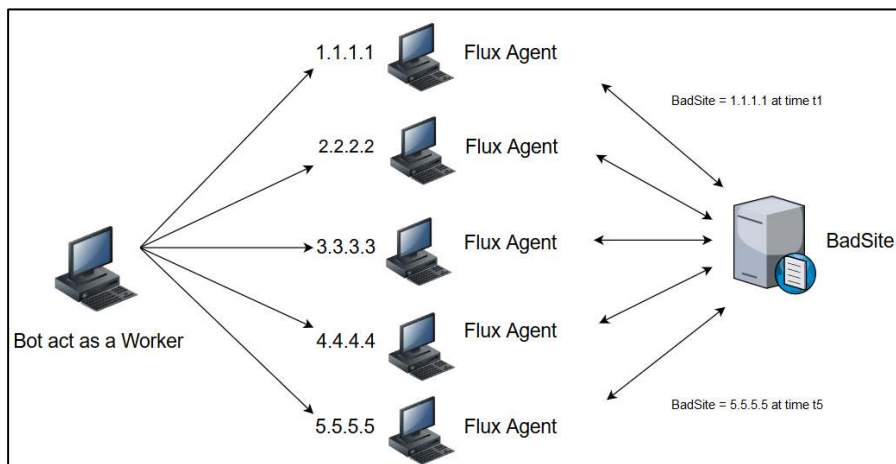


Figure 2: Fast-Flux service network.

Using FFSN, a domain name resolves different IP addresses depending on the exact time in which the petition is made, which enables the decentralization of the C&C servers and complicates unraveling the structure of the botnet. To carry out the IP resolution changes, these domains have very low TTL in the cache, which forces the DNS systems to frequently refresh the resolution cache of the IP addresses associated to the domain. In the case of a null TTL, the resolution is not even stored. Therefore, those DNS petitions whose TTL is low are suspicious (Cantón, 2015).

Table 3 illustrates the Concept of Domain Flux.

Table 3

Domain Flux Implementation

Time	IP Address	Domain Name
T1	1.1.1.1 3.3.3.3 7.7.7.7	botnet.com
T2	1.1.1.1 3.3.3.3 7.7.7.7	malicious.com
T3	1.1.1.1 3.3.3.3 7.7.7.7	C&C.com
T4	1.1.1.1 3.3.3.3 7.7.7.7	suspicious.com

In DF, the botnets evade the detection by implementing short TTL periods and cycling of IP mappings for the domain name of C&C-servers (Lysenko et al., 2015).

Table 4 illustrates the concept of FFSN and DF combination. The DF can be implemented along with DGA or can be utilized by instructing bot(s)–already connected–to request different domains next time.

Table 4

FFSN & DF Implementation

Time	IP Address	Domain Name
T1	1.1.1.1 5.5.5.5	botnet.com
T2	3.3.3.3 7.7.7.7	malicious.com
T3	2.2.2.2 9.9.9.9	C&C.com
T4	3.3.3.3 7.7.7.7	botnet.com
T5	1.1.1.1 5.5.5.5	C&C.com
T6	2.2.2.2 9.9.9.9	malicious.com
T7	3.3.3.3 7.7.7.7	C&C.com
T8	2.2.2.2 9.9.9.9	botnet.com
T9	1.1.1.1 5.5.5.5	malicious.com
T10	3.3.3.3 7.7.7.7	malicious.com
T11	2.2.2.2 9.9.9.9	botnet.com
T12	3.3.3.3 7.7.7.7	C&C.com

In DNS tunneling, a botmaster can abuse the Domain Name Service protocol, if the DNS traffic is not restricted, to establish a C&C channel between the bot(s) and the C&C server. These channels are difficult to detect and block. Some DNS tunneling tools support SSH, such as Iodine and DNS2TCP. These tools can be

utilized to encrypt botnet communications to complicate traffic inspection. Figure 3 shows how a bot communicates with a C&C server through DNS tunnel.

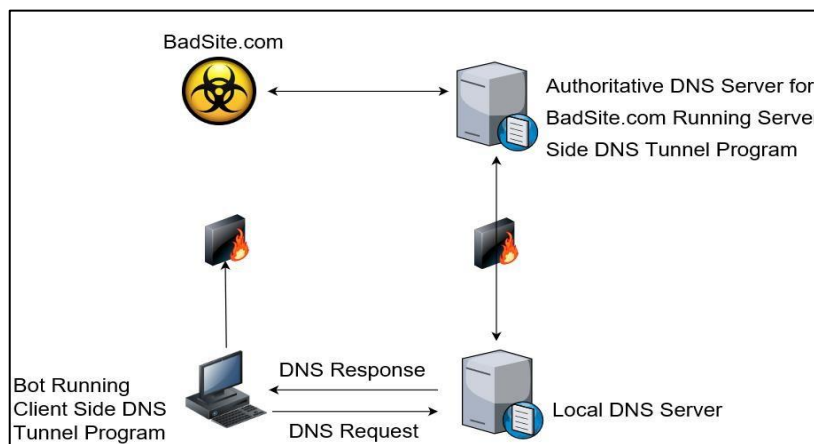


Figure 3. DNS tunneling.

Literature Review

Dynamic domain generation, fluxing, and tunneling techniques have been used by different malware families to avoid detection and complicate mitigation efforts. Research communities have proposed many different approaches and mechanisms for the development of the botnet-detection techniques. Many of these approaches are effective for specific types of botnet.

Bilge, Sen, Balzarotti, Kirda, and Kruegel (2011) identified a feature vector with 15 different features for malicious domain detection. These features are classified into four feature sets:

1. Time-Based Features.
2. DNS Answer-Based Features.
3. TTL Value-Based Features.
4. Domain Name-Based Features.

Table 5 shows that each feature set has a different feature for malicious domain detection.

Table 5

Feature Vector (Bilge et al., 2011)

Feature Set	Feature Name
Time-Based Feature	Short life Daily Similarity Repeating patterns Access ratio
DNS Answer-Based Features	Number of distinct IP addresses Number of distinct countries Number of domains share the IP with Reverse DNS query results
TTL Value-Based Features	Average TTL - Standard Deviation of TTL - Number of distinct TTL values - Number of TTL change - Percentage usage of specific TTL ranges
Domain Name-Based Features	- Percentages of numerical characters - Percentage of the length of the LMS

Krmíček (2011) examined the NetFlow¹ of DNS IP traffic and its relation to the botnet presence in the monitored network. He studied the DNS behavior of known malicious and benign domains based on features identified by Bilge, Sen, Balzarotti, Kirida, and Kruegel (2011). Since NetFlow inspects only packet headers, not the

¹ Unidirectional sequence of packets with some common properties that pass through a network device. (p. 1).

entire packet payload, Krmíček concluded that "using NetFlow data solely, for the purpose of botnet detection is not possible" (p. 8), and he mentioned that extracting important information from the packet payload is the most promising approach for botnet detection.

Choi, Lee, and Kim (2007) proposed a botnet detection mechanism by monitoring DNS traffic, which forms a group activity in DNS requests simultaneously sent by many distributed bots. Upon successful infection, the bots rally to a C&C server at an early stage. In other words, the bots will have to register with the C&C server. If the IP address of the C&C server is not hard coded, the bots use DNS in a rallying process, and the DNS traffic has unique features defined as group activity (Domain Names & Timestamps). Their mechanism uses the information of IP headers to detect botnets, irrespective of the protocol used.

Choi et al. (2007) developed a mechanism to detect C&C server migration, where a botnet frequently changes its C&C server—to avoid dismantling—by migrating to a candidate C&C server using DDNS. The authors summarized the differences between botnet DNS traffic and legitimate DNS traffic in Table 6.

Table 6

Differences between Botnet and Legitimate DNS

	Source IPs accessed to domain name	Activity	Appearance pattern	DNS Type
Botnet DNS	Fixed size (Botnet members)	Group activity	Intermittently	Usually DDNS
Legitimate DNS	Anonymous (Legitimate users)	Non-group activity	Randomly and continuously	Usually DNS

There are some limitations to this mechanism: monitoring a huge scale of networks poses high processing times and presents significant problems. Also, their algorithms can be evaded when the botnet uses DNS only at initializing and never again (moreover, do not migrate the botnet). Furthermore, since their mechanism is based on the similarity of group activity, this makes it not suitable for detecting small numbers of infected machines in a monitored network.

Dietrich et al. (2011) are the first to document DNS-based botnet C&C traffic. They presented a technique for DNS-based C&C traffic detection and another technique for malware sample classification based on their behavior. Their work is based on the high entropy of C&C messages generated by Feederbots; they utilized the fact that encrypted or compressed messages have high entropy.

A limitation of their technique is that, for certain resource records, the distribution of byte values could be compared against the expected distribution (e.g., rdata of A RR contains IPv4 addresses). However, the IPv4 address space is not uniformly distributed (e.g., reserved addresses, such as private addresses or multicast addresses, might rarely show up in Internet DNS traffic), whereas other addresses, such as popular websites, might appear more often in DNS query results. Another limitation is that:

Botmasters could restrict their C&C messages to very small sizes. In practice, message content could be stored in, e.g., 4 bytes of an A resource record's rdata. In this case, our rdata features alone, which are applied to individual C&C message would not be able to detect these C&C messages as high

entropy messages because the statistical byte entropy of really short messages is very low, and our estimate of the alphabet size by counting the number of distinct byte is inaccurate for very short messages. (p. 15)

Lysenko et al. (2015) proposed a DNS-based anti-evasion technique for botnet detection. Their technique is based on a cluster analysis of the features obtained from the payload of incoming DNS messages. The method uses the semi-supervised fuzzy c-means clustering. Figure 4 illustrates the functioning principle of the DNS-based anti-evasion technique for botnet detection.

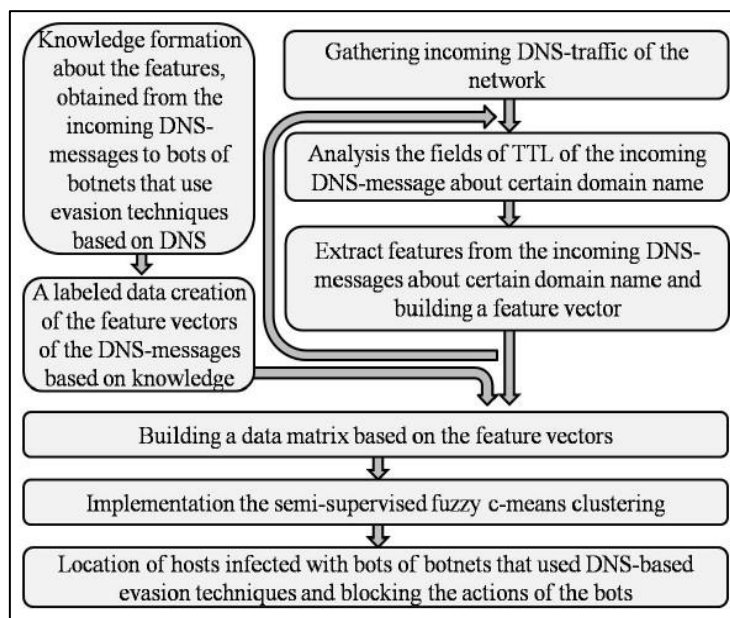


Figure 4. Functioning principle of the DNS-based anti-evasion technique for botnet detection (Lysenko et al., 2015).

According to Lysenko et al. (2015), their technique can detect fast flux, domain flux, cycling of IP mappings, and DNS tunneling evasion techniques with high efficiency. They claimed that passive analysis of DNS traffic leads to the detection of only particular malware.

Jin, Ichise, and Iida (2015) designed a botnet communication detection method by collecting authoritative NS records and their IP addresses, as well as monitoring direct outbound DNS queries. Their method is based on storing NS records with corresponding IP addresses of valid query response pairs, IP addresses of public DNS servers, and ISP specified DNS servers in a NS-IP database. Any destination IP address is not included in the previously achieved Name Server NS records, as well as its corresponding IP Address; a record is considered suspicious and should be investigated. In this way, "all unusual domain name resolution that uses direct outbound DNS query can be monitored" (p. 39).

A DNS tunneling technique could evade their method. Domain Name System tunneling can be used for a more robust C&C configuration. For example, a botmaster could register the malicious domain name and designate the system running dnscat2 server software as the authoritative DNS server for that domain. In this way, the bot machine would issue a DNS query for that malicious domain to the victim's trusted DNS server, which would forward the query to the C&C server and return the adversary's answer to the bot. In this scenario, the protected network can only access the trusted DNS server, but that DNS server can contact external DNS servers to resolve queries that it cannot resolve directly (Zeltser, 2016). Since the returned malicious answer is from an authoritative DNS server, it would be stored in the NS-IP whitelist database, resulting in false-negative alert.

Holz, Gorecki, Rieck, and Freiling (2008) presented the first empirical study of FFSNs. They developed a metric that exploits the principles of FFSNs to derive an

effective mechanism for detecting new fast-flux domains in an automated way. They showed that the method is accurate, and they had very low false-positive and false-negative rates. Based on their empirical observations, they found other information (e.g., whois lookups and MX records) as promising features for an extended version of their flux-score.

Caglayan, Toothaker, Drapeau, Burke, and Eaton (2009) presented the first empirical study of detecting and classifying fast flux service networks (FFSNs) in real time. Their approach uses active and passive sensors derived from DNS monitoring and fusing the component sensors using a Bayesian classifier. The Fast Flux Monitor Architecture can detect single and double flux behavior in real time with acceptable false alarm rates.

Dabbagh (2012) proposed a method for detecting IP ID and TTL covert channels. He proposed a method based on his observation that "operating systems choose initially a random number for the ID in the IP header and then increment it sequentially" (p. 1). He concluded that a packet is suspicious if the new packet has an IP ID smaller than the previous packets. Also, he stated that "detecting TTL covert channel is based on the fact that the network is stable" (p. 2). Therefore, the receiver side should not observe many variations in the TTL values in the IP header of the packets that are coming from the same source.

A limitation of this method is when using NAT services, packets coming from different sources will have different IDs and TTL values, but will have the same source IP. Another limitation is that some IP stacks assign the ID values of the IP

header by using a pseudo-random generator (RFC 4413-TCP/IP Field Behavior, 2006).

Zhang, Papadopoulos, and Massey (2013) made an initial attempt to investigate detection of encrypted communication. Since the encryption increases entropy, they presented two high-entropy classifiers and used one of them to enhance the BotHunter, and showed that BotHunter was able to detect encrypted bots.

Antonakakis et al. (2012) presented a novel detection system, called Pleiades, which is able to detect machines within a monitored network that are compromised with DGA-based botnets. Pleiades monitors traffic below the local recursive DNS server and analyzes streams of unsuccessful DNS resolutions (Name Error or NXDomain Responses). Pleiades searches for relatively large clusters of NXDomains with similar syntactic features, and are queried by multiple, potentially compromised, machines during a specific epoch. As shown in Figure 5, there are two phases of detection: the first phase discovers the presence of DGA and the second classifies the discovered DGA and detects the C&C domain(s).

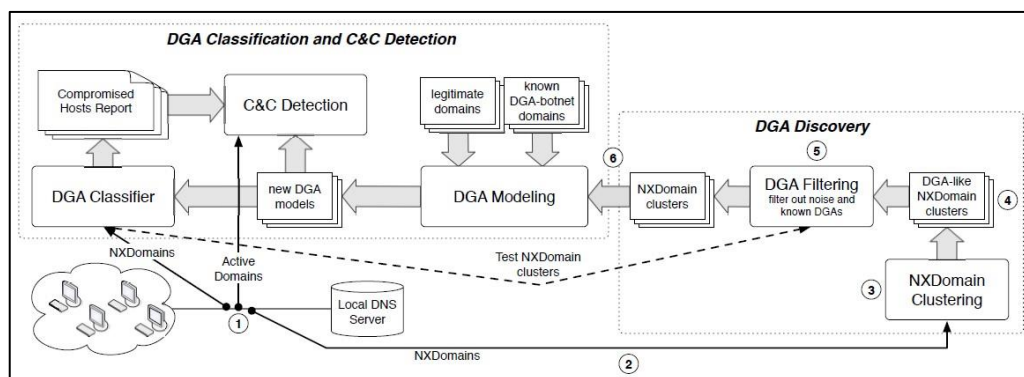


Figure 5. A High-Level Overview of Pleiades (Antonakakis et al., 2012).

Antonakakis et al. (2012) claimed that "Pleiades was able to identify six DGAs that belong to known malware families and six new DGAs never reported before" (p. 14). Although their claim that Pleiades can achieve very high detection accuracy, one limitation of their evaluation method is "the exact enumeration of the number of infected hosts in the ISP network" (p. 14). Because the location of monitoring sensors is below the recursive DNS server, they can only obtain a lower bound estimate of infected hosts. For example, an IP address that generates DNS traffic may be a NAT, firewall, DNS server, or other device that behaves as a proxy. Also, noisy NXDomains may be generated to mislead the implementation of Pleiades.

Yadav and Reddy (2012) proposed methodologies for utilizing failed domain names (NXDOMAIN) in the quest for rapid detection of a fluxing botnet's C&C server. They validated their method by detecting Conficker botnets and other anomalies with a false positive rate as low as 0.02%. Their technique can be applied at the edge of an autonomous system for real-time detection. Since their method is based on detecting botnets utilizing high entropy, botnet owners may alter the way domain names are created to evade their detection mechanism.

Farnham and Atlasis (2013) reviewed several utilities used to enable tunneling over DNS. They discussed practical techniques for detecting DNS tunneling and categorized the detection techniques into two categories: payload detection technique, which is used to detect specific DNS tunneling utilities, and traffic analysis-based technique, which is used to detect DNS tunneling in general. In the

payload analysis, they discussed the following techniques for DNS tunneling detection:

1. Size of DNS request and response.
2. Entropy of hostnames.
3. Statistical analysis.
4. Uncommon record types.
5. Policy violation.
6. Specific signatures.

In traffic analysis, they discussed the following techniques:

1. Volume of DNS traffic per IP address.
2. Volume of DNS traffic per domain.
3. Number of hostnames per domain.
4. Geographical location of DNS server.
5. Domain history.
6. Volume of NXDomain responses.
7. Visualization.
8. Orphan DNS requests.
9. General covert channel detection.

In this research, the NXDomain error will be utilized in a different way to detect the DGA implementations; the threshold relies on the percentages of the unique NXDomain errors to the total number DNS requests within an epoch. Although the DNS Server Failure error is not limited to the FFSN and DF implementations, this

error can be utilized to detect FFSN- and DF-based botnets. In other words, the first stage will utilize NXDomain and Server Failure errors to detect the rallying to a C&C server.

In the second phase, although polymorphism and metamorphism techniques change the form of each instance of bot binary to circumvent signature-based detection during the detection and investigative process, the algorithm will use a detection technique relying on a signature matching based on encoded SSH handshakes within DNS tunnels.

Summary

This chapter presented an overview of botnets implementation, as well as some detection methods and evasion techniques. The next chapter proposes a defense-in-depth approach for DNS-based botnets.

Chapter III: Methodology

Introduction

Implementing a comprehensive, holistic approach for botnet detection could be a challenging task. Botnets implement different protocols, different architecture, and can evade detection methods by tunneling their communications within a range of services, DNS being the most predominant.

Since the DNS protocol can be used at different stages of botnet communication, I have used BRO Network Security Monitor (NSM) to design and implement a detection mechanism for DNS-based botnets communication.

Design of the Study

This thesis proposes an empirical solution to design and implements a mechanism for detecting DNS-based botnets at different stages:

1. Rallying stage when finding the C&C server (DGA, FFSN, and DF).
2. Transmitting data and controlling the bots (DNS Tunnel).

Currently, botnets implement DGA and/or fluxing techniques to avoid botnets detection and mitigation. The infected machine sends a high volume of DNS requests in order to find its C&C server.

As a botmaster only registers one or a few domain names (previously known) to carry out the C&C communication, almost all the DNS requests, generated by DGA, sent to find the C&C server will have unsuccessful resolutions (name error or NXDomain responses). The detection of DGA implementation was configured based on a threshold of NXDomain responses within an epoch. For example, if the infected

machine sends more than 100 DNS requests within an hour, and a specific percentage of these requests have unsuccessful resolutions, BRO NSM will detect the presence of DGA based botnet.

In FFSN and DF based botnets, the malicious domain name(s) that has/have very low TTL forces the DNS systems to frequently refresh the resolution cache of the IP addresses associated with the domain(s). Although the DNS Server Failure can be related to issues other than fluxing implementations, these unsuccessful resolutions of very low TTL domain names (Server Failure) can be utilized for FFSN- and DF-based botnets detection. In other words, the detection of FFSN and DF implementation was configured based on a threshold of the "Server Failure" responses within an epoch.

The frequency of malicious DNS packets can be controlled by the botmaster to evade the detection threshold. In other words, in case the first BRO mechanism fails to detect DGA, FFSN, or DF presence, another mechanism will run. The second mechanism inspects the DNS payloads for DNS-encrypted tunnels based on SSH connections, also implemented with BRO NSM.

According to Brandhorst and Pras (2006) on their statistical analysis of name server traffic, the percentages of NXDomain errors and Server Failures were 8.74% and 1.28%, respectively, of the DNS queries. Figure 6 shows DNS statistics at four locations.

	Location #1	Location 2	Location 3	Location 4	All
Date	2002/05/23 – 06/26	2003/05/13 – 08/28	2003/09/02 – 11/25	2004/02/04 – 05/07	
Total Gigabytes	130.7	107.9	885.6	1,230.7	2,354.9
DNS Megabytes (1)	29.8 (0.02%)	285.4 (0.26%)	719.7 (0.08%)	75.4 (0.01%)	1,110.2 (0.05%)
Total packets	220,314,110	167,772,874	1,346,774,765	2,141,051,358	3,875,913,107
DNS packets (2)	228,883 (0.10%)	2,365,875 (1.41%)	5,885,995 (0.44%)	642,427 (0.03%)	9,123,180 (0.24%)
Queries (3)	115,679 (50.54%)	1,280,706 (54.13%)	3,130,427 (53.18%)	336,207 (52.33%)	4,863,019 (53.30%)
Recursive (4)	112,825 (97.53%)	11,011 (0.86%)	149,367 (4.77%)	180,959 (53.82%)	454,162 (9.34%)
Iterative (4)	2,854 (2.47%)	1,269,695 (99.14%)	2,981,060 (95.23%)	155,248 (46.18%)	4,408,857 (90.66%)
Unanswered (4)	2,475 (2.14%)	195,537 (15.27%)	374,859 (11.97%)	29,987 (8.92%)	602,858 (12.40%)
OK (4)	72,308 (62.51%)	868,755 (67.83%)	2,404,557 (76.81%)	208,464 (62.00%)	3,554,084 (73.08%)
Format error (4)	291 (0.25%)	58,050 (4.53%)	90,234 (2.88%)	18,775 (5.58%)	167,350 (3.44%)
Server error (4)	1,761 (1.52%)	28,619 (2.23%)	28,181 (0.90%)	3,874 (1.15%)	62,435 (1.28%)
No such name (4)	38,534 (33.31%)	127,214 (9.93%)	225,566 (7.21%)	33,650 (10.01%)	424,964 (8.74%)
Not implemented (4)	118 (0.10%)	75 (0.01%)	2,348 (0.08%)	1,448 (0.43%)	3,989 (0.08%)
Refused (4)	192 (0.17%)	2,456 (0.19%)	4,682 (0.15%)	40,009 (11.90%)	47,339 (0.97%)
Average latency (ms)	919, $\sigma = 399$	114, $\sigma = 64$	85, $\sigma = 38$	332, $\sigma = 344$	152, $\sigma = 205$

*Corrected trace statistics. Percentages are with respect to: (1) Total bytes; (2) Total # of packets; (3) # of DNS packets; (4) # of Queries.

Figure 6. DNS statistics (Brandhorst & Pras, 2006).

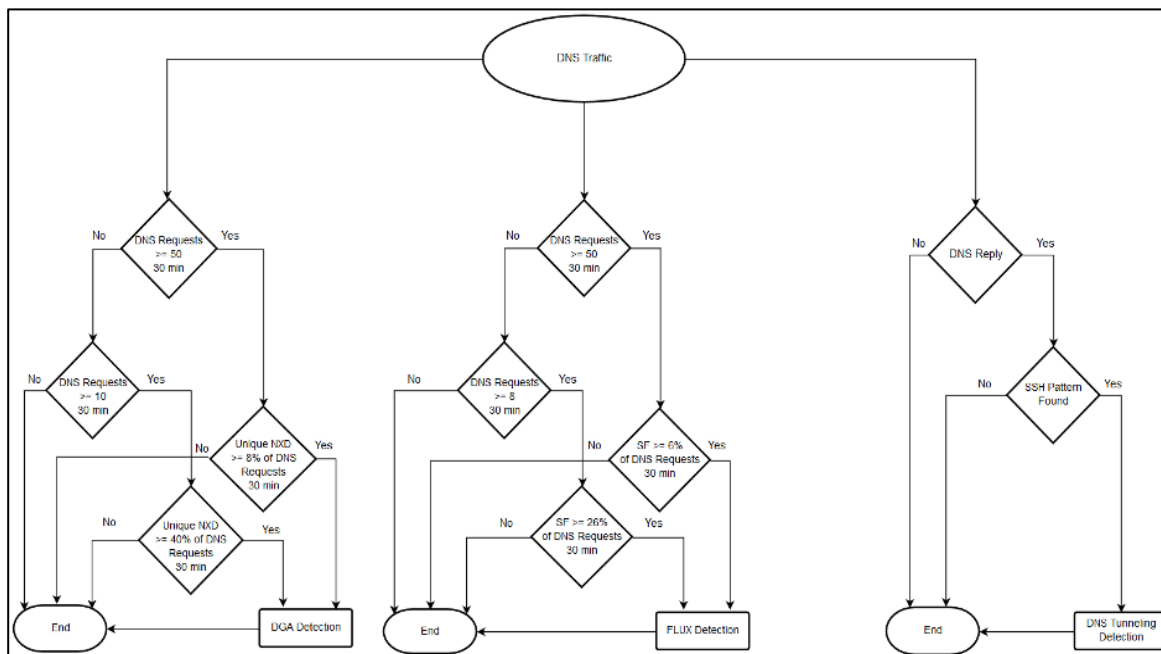


Figure 7. Multistage detection technique for DNS-based botnets.¹

¹ NXD refers to NXDomain errors and SF refers to Server Failure errors.

Figure 7 depicts the flowchart for the proposed algorithm. Domain Name System traffic is inspected at both stages, in parallel, to detect the multistage communications of DNS-based botnets. The first stage detects the rallying to a C&C server and the second stage inspects each DNS reply of specific patterns of SSH handshakes within DNS tunnels.

In the first stage of botnet communication (rallying stage), a DGA-detection mechanism is applied every 30 minutes to find if total number of DNS requests is 10 or greater. In this case, two thresholds are used for the unique NXDomain errors: the first threshold, which is 8% of the total number of DNS requests, is based on Figure 6 and used if the total number of DNS requests is 50 or more. The second threshold (40%) is used if the total number of DNS requests is between 10 and 50. The second threshold is utilized to eliminate false positive detections in idle systems. For example, an idle system running DGA has a higher percentage of unique NXDomain errors to the total number of DNS requests than the percentage in an active system running DGA. In both thresholds, at least four unique NXDomain errors are required for DGA detection.

In fluxing detection, which is another method for rallying, different thresholds are set. The fluxing detection mechanism is applied every 30 minutes if the total number of DNS requests is 8 or greater. In this mechanism, two thresholds are used for the Server Failure errors. If the total number of DNS requests is 50 or greater, the first threshold is used, which is 6% of the total number of DNS requests. The second threshold (26%) is used if the total number of DNS requests is between 8 and 50.

Similar to the DGA mechanism, the second threshold is utilized to eliminate false-positive detection in idle systems implementing fluxing techniques. In both thresholds, at least three Server Failure errors are required for fluxing detection.

The DNS threshold, which is 50 DNS requests every 30 minutes, is set below the lowest average in Figure 6 (location 1 has an average of 137 DNS requests per hour). Since location 1 in Figure 6 has a high percentage of NXDomain errors (33%), which indicates a high possibility of DGA existence, the NXDomain threshold (8%) is set based on the lowest average of NXDomain errors (location 3). To eliminate false-negative detection, the Server Failure threshold (6%) is set higher than the Server Failure percentage in location 2.

In other words, different threshold values are set for DGA and fluxing detection, based on the DNS statistics in Figure 6, as well as the activity of the infected systems. For example, the percentage of NXDomain errors to the total number of DNS requests depends on the frequency of DGA and the average number of DNS requests on different systems. Thus, these values need to be dynamically adjusted with the changing nature of communication.

Data Collection

Instead of real-time monitoring with real botnet malware, the dataset was collected from the Stratosphere Lab, which is a part of the Malware Capture Facility Project at CVUT University, Prague, Czech Republic (Garcia, 2015). The lab has a significant dataset of malware traffic captures, including different types of botnets. These datasets were used for DGA, FFSN, and DF detection.

Regarding DNS encrypted tunnels, Iodine and DNS2TCP were used to setup a SSH tunnel between the server and client device. The captured traffic was used for SSH connection detection.

Data Analysis

In order to implement and analyze the proposed approach, the following software, tools, and techniques were required:

1. BRO Network Security Monitor: intrusion detection system.
2. Iodine & DNS2TCP: DNS tunneling tools that support SSH.
3. Wireshark: traffic analyzer.
4. Security Onion: a customized Linux operating system for intrusion detection.
5. AWS Ubuntu machines: C&C server and infected machine.
6. Registered Domain Name.

Summary

This chapter covered the proposed detection methodology for DNS-based botnets. This methodology is built on a BRO Network Security Monitor to detect both DGA- and FFSN-based botnets according to thresholds within an epoch of NXDomain and Server Failure responses, respectively. Also, the proposed methodology detects DNS-encrypted tunnels through analyzing connection establishment within a DNS payload. The next chapter presents more detail about data collection and analysis.

Chapter IV: Data Presentation and Analysis

Introduction

Different malicious traffic captures were collected from previously infected systems. These captures were used for DGA-, FFSN-, and DF-based botnet analysis. For SSH connections tunneled in DNS packets, Iodine and DNS2TCP (with SSH connections) captures were used for packet analysis.

Data Presentation

In this section, the Wireshark packet captures were presented as the following:

1. **Packet capture of DGA-based botnet.** This capture was collected from a previously infected system and was used to test the BRO detection method for DGA-based botnets. The complete capture can be found at [Wireshark–DGA-based Botnet](#).
2. **Packet capture of FFSN- & DF-based botnet.** This capture was collected from a previously infected system and was used to test the BRO detection method for FFSN- and DF-based botnets. The complete capture can be found at [Wireshark–FFSN- & DF-based Botnet](#).
3. **Packet capture of SSH connection tunneled in Iodine.** Iodine is a DNS tunneling program that tunnels IPv4 data through a DNS server. It was developed by Bjorn Anderson and Erik Ekman. Iodine can be usable when the Internet access is firewalled, but DNS queries are allowed. Iodine is written in C and it runs on Linux, Mac OS X, FreeBSD, NetBSD, OpenBSD, and Windows (Anderson & Ekman, 2014).

The packet capture of Iodine traffic was collected from the lab implementation. In this lab, the Iodine server and Iodine client were setup as the following:

- Iodine server static IP address 52.52.65.253
- Iodine client IP address 172.31.36.28
- Subdomain tunnel.ialabs.net
- Server's tunnel interface IP address 192.168.250.1
- Client's tunnel interface IP address 192.168.250.2

The complete lab steps can be found in Appendix B.

4. **Packet capture of SSH connection tunneled in DNS2TCP.** DNS2TCP is a network tool designed to relay TCP connections through DNS traffic. DNS2TCP was written by Olivier Dembour with the contributions of Nicolas Collignon. The encapsulation is done on the TCP level. DNS2TCP is composed of two parts: a server-side tool and a client-side tool. The server has a list of resources specified in a configuration file. Each resource is a local or remote service listening for TCP connections. The client listens on a predefined TCP port and relays each incoming connection through DNS to the final service (HSC, 2012).

The packet capture of DNS2TCP traffic was also collected from the lab implementation. In this lab, the two sides—the server and the client—were setup as the following:

- DNS2TCP server static IP address 52.52.65.253
- DNS2TCP client IP address 172.31.36.28
- Subdomain tunnel2.ialabs.net

The complete lab steps can be found in Appendix C.

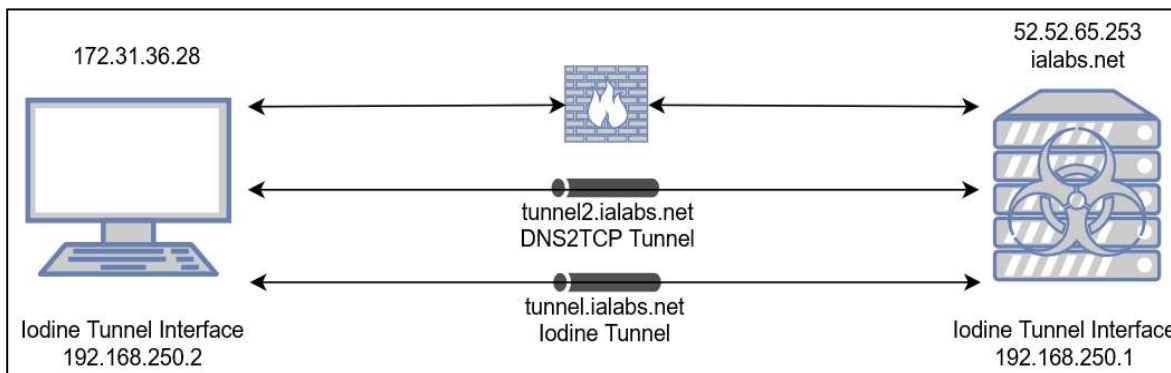


Figure 8. Iodine and DNS2TCP lab.

Figure 8 shows the lab implementation of Iodine and DNS2TCP tunneling tools. The system is protected by a firewall that blocks all traffic except DNS on port UDP/53.

Data Analysis

The packet captures of the previous section were analyzed as the following:

1. **Packet capture of DGA implementation.** Figure 9 shows the statistics of DNS packets. The "No such name" packets represent the total number of DNS responses that return with a NXDomain error. There were 352,756 responses with NXDomain error; these responses represented 46.75% of the total DNS packets (queries and responses), and 89% of the total requests (532,756/398,397). The "No error" packets represent both the total number of DNS requests and the successful responses.

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
▼ Total Packets	754601				0.0002	100%	0.0400	416309.664
▼ rcode	754601				0.0002	100.00%	0.0400	416309.664
Server failure	7				0.0000	0.00%	0.0100	572.960
No such name	352756				0.0001	46.75%	0.0100	2244226.151
No error	401838				0.0001	53.25%	0.0400	416309.664
▼ opcodes	754601				0.0002	100.00%	0.0400	416309.664
Standard query	754601				0.0002	100.00%	0.0400	416309.664
▼ Query/Response	754601				0.0002	100.00%	0.0400	416309.664
Response	356204				0.0001	47.20%	0.0200	13.152
Query	398397				0.0001	52.80%	0.0300	2257257.001
▼ Query Type	754601				0.0002	100.00%	0.0400	416309.664
AAAA (IPv6 Address)	52				0.0000	0.01%	0.0200	584.337
A (Host Address)	754549				0.0002	99.99%	0.0400	2453613.072
▼ Class	754601				0.0002	100.00%	0.0400	416309.664
IN	754601				0.0002	100.00%	0.0400	416309.664

Figure 9. DNS packets statistics–DGA-based botnet.

The previous capture shows a high presence of DGA implementation. Setting a threshold for DGA detection varies from one network to another. For example, when implementing a DGA botnet, botmasters can control the frequency of DNS requests to avoid detection, thus, a high threshold of NXDomain errors can be evaded by a low frequency (false negative), however, a very low threshold may result in false-positive detection, such as typing errors. To eliminate the chance of NXDomain errors due to mistyping or other errors not related to the DGA implementation, the threshold was set based on unique NXDomain records.

Figure 10 shows DGA-based botnet traffic. After many unsuccessful resolutions, the infected system found the IP address of a C&C server at packet number 234,427.

2013-08-20_capture-win1.pcap

No.	Time	Source	Destination	Protocol	Length	Info
232420	2411722.457429	10.0.2.15	8.8.8.8	DNS	88	Standard query 0xa92b A hxxkjbihqcciihohyplrtdq.net
232421	2411722.504797	8.8.8.8	10.0.2.15	DNS	161	Standard query response 0xa92b No such name A hxxkjbihqcciihohyplrtdq.net SOA a.gtld-servers.net
232422	2411723.988383	10.0.2.15	8.8.8.8	DNS	90	Standard query 0x8f66 A lsvpgttbgpylrweuocynfqs1q.org
232423	2411724.033988	8.8.8.8	10.0.2.15	DNS	153	Standard query response 0x8f66 No such name A lsvpgttbgpylrweuocynfqs1q.org SOA a0.org.afillias-nst.info
232424	2411725.501264	10.0.2.15	8.8.8.8	DNS	93	Standard query 0x329a A jfwlnrcivcckpaduvouaguvlgeib.biz
232425	2411725.547175	8.8.8.8	10.0.2.15	DNS	155	Standard query response 0x329a No such name A jfwlnrcivcckpaduvouaguvlgeib.biz SOA a.gtld.biz
232426	2411727.019093	10.0.2.15	8.8.8.8	DNS	90	Standard query 0x0028 A orhylvbiyteikneydhirbuxxo.com
232427	2411727.064908	8.8.8.8	10.0.2.15	DNS	106	Standard query response 0x0028 A orhylvbiyteikneydhirbuxxo.com A 69.163.37.12
232428	2411727.065404	10.0.2.15	69.163.37.12	TCP	62	58644 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
232429	2411730.032831	10.0.2.15	69.163.37.12	TCP	62	[TCP Retransmission] 58644 → 80 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
232430	2411734.280296	69.163.37.12	10.0.2.15	TCP	58	80 → 58644 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
232431	2411734.280491	10.0.2.15	69.163.37.12	TCP	54	58644 → 80 [ACK] Seq=1 Ack=1 Win=64240 Len=0
232432	2411734.280787	10.0.2.15	69.163.37.12	HTTP	331	GET / HTTP/1.1
232433	2411734.280861	69.163.37.12	10.0.2.15	TCP	54	80 → 58644 [ACK] Seq=1 Ack=278 Win=65535 Len=0
232434	2411769.862794	10.0.2.15	69.163.37.12	TCP	54	58644 → 80 [FIN, ACK] Seq=278 Ack=1 Win=64240 Len=0
232435	2411769.862931	69.163.37.12	10.0.2.15	TCP	54	80 → 58644 [ACK] Seq=1 Ack=279 Win=65535 Len=0
232436	2411770.063573	69.163.37.12	10.0.2.15	TCP	54	80 → 58644 [FIN, ACK] Seq=1 Ack=279 Win=65535 Len=0
232437	2411770.063747	10.0.2.15	69.163.37.12	TCP	54	58644 → 80 [ACK] Seq=279 Ack=2 Win=64240 Len=0
232438	2411771.348528	10.0.2.15	8.8.8.8	DNS	87	Standard query 0x0eeb A jizslrcojvcnfonjrtotkdxs.ru
232439	2411771.395827	8.8.8.8	10.0.2.15	DNS	148	Standard query response 0x0eeb No such name A jizslrcojvcnfonjrtotkdxs.ru SOA a.dns.ripn.net
232440	2411772.871955	10.0.2.15	8.8.8.8	DNS	89	Standard query 0xe3a6 A vk1bsgtwrfjwkkzjlfstvibvo.com
232441	2411772.919009	8.8.8.8	10.0.2.15	DNS	162	Standard query response 0xe3a6 No such name A vk1bsgtwrfjwkkzjlfstvibvo.com SOA a.gtld-servers.net

Figure 10. Wireshark packet capture of DGA-based botnet.

2. **Packet capture of FFSN & DF implementation.** Similar to the previous statistics, Figure 11 shows the statistics of DNS packets. The "Server Failure" packets represent the total number of DNS responses that return a Server Failure error. There were 159 responses with Server Failure errors; these responses represented 26.07% of the total DNS packets (queries and responses), and 52.13% of the total requests (159/305).

Wireshark - DNS - 2015-07-08_capture-win8

Topic / Item	Count	Average	Min val	Max val	Rate (ms)	Percent	Burst rate	Burst start
Total Packets	610				0.0000	100%	0.0400	88.274
rcode	610				0.0000	100.00%	0.0400	88.274
Server failure	159				0.0000	26.07%	0.0200	172995.087
No error	451				0.0000	73.93%	0.0400	88.274
opcodes	610				0.0000	100.00%	0.0400	88.274
Standard query	610				0.0000	100.00%	0.0400	88.274
Query/Response	610				0.0000	100.00%	0.0400	88.274
Response	305				0.0000	50.00%	0.0200	88.301
Query	305				0.0000	50.00%	0.0200	88.274
Query Type	610				0.0000	100.00%	0.0400	88.274
AAAA (IPv6 Address)	10				0.0000	1.64%	0.0200	117.353
A (Host Address)	600				0.0000	98.36%	0.0400	88.274
Class	610				0.0000	100.00%	0.0400	88.274
IN	610				0.0000	100.00%	0.0400	88.274
Response Stats	0				0.0000	100%	-	-

Display filter: Enter a display filter ...

Apply Copy Save as... Close

Figure 11. DNS packets statistics—FFSN & DF-based botnet.

This capture shows a high percentage of DNS Server Failure packets, and setting a threshold for these packets is also a challenge. For example, botmasters can control the TTL values and the frequency of DNS requests to avoid Server Failure errors, thus, a high threshold of Server Failure errors can be evaded by a lower frequency and higher TTL (false negative). However, a low threshold may result in a higher false positive.

Figure 12 shows the DF implementation. There were three IP addresses (87.221.209.204, 109.73.179.95, and 185.1.62.82) assigned to four different malicious domains (top-web.org, linetechservice.org, serviceline2013.org, and servicewebcheck.org); the assignment was done in a “round robin” fashion. When the DNS request (packet No. 15680) was sent, the IP addresses were not yet assigned to serviceonline2013.org, thus, packet No. 15683 had a Service Failure error since the TTL for serviceonline2013.org had expired.

No.	Time	Source	Destination	Protocol	Length	Info
15672	86586.468570	10.0.2.108	8.8.8.8	DNS	71	Standard query 0xf5f7 A top-web.org
15673	86586.485981	8.8.8.8	10.0.2.108	DNS	119	Standard query response 0xf5f7 A top-web.org A 185.1.62.82 A 109.73.179.95 A 87.221.209.204
15674	86587.585937	10.0.2.108	8.8.8.8	DNS	79	Standard query 0x35c1 A linetechservice.org
15675	86587.624949	10.0.2.108	85.17.87.163	TCP	66	[TCP Retransmission] 51309 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 WS=256 SACK_PERM=1
15676	86588.585925	10.0.2.108	8.8.4.4	DNS	79	Standard query 0x35c1 A linetechservice.org
15677	86588.605340	8.8.4.4	10.0.2.108	DNS	127	Standard query response 0x35c1 A linetechservice.org A 185.1.62.82 A 87.221.209.204 A 109.73.179.95
15678	86589.763964	8.8.8.8	10.0.2.108	DNS	127	Standard query response 0x35c1 A linetechservice.org A 109.73.179.95 A 185.1.62.82 A 87.221.209.204
15679	86589.764215	10.0.2.108	8.8.8.8	ICMP	155	Destination unreachable (Port unreachable)
15680	86590.800709	10.0.2.108	8.8.8.8	DNS	81	Standard query 0x9d6e A serviceonline2013.org
15681	86591.800570	10.0.2.108	8.8.4.4	DNS	81	Standard query 0x9d6e A serviceonline2013.org
15682	86592.194599	8.8.4.4	10.0.2.108	DNS	129	Standard query response 0x9d6e A serviceonline2013.org A 185.1.62.82 A 109.73.179.95 A 87.221.209.204
15683	86592.828926	8.8.8.8	10.0.2.108	DNS	81	Standard query response 0x9d6e Server failure A serviceonline2013.org
15684	86592.829194	10.0.2.108	8.8.8.8	ICMP	109	Destination unreachable (Port unreachable)
15685	86593.623631	10.0.2.108	85.17.87.163	TCP	62	[TCP Retransmission] 51309 → 443 [SYN] Seq=0 Win=8192 Len=0 MSS=1460 SACK_PERM=1
15686	86594.485334	10.0.2.108	185.48.56.166	TCP	54	51305 → 80 [RST, ACK] Seq=6106 Ack=859 Win=0 Len=0
15687	86594.946733	10.0.2.108	8.8.8.8	DNS	79	Standard query 0xb503 A servicewebcheck.org
15688	86594.965974	8.8.8.8	10.0.2.108	DNS	127	Standard query response 0xb503 A servicewebcheck.org A 185.1.62.82 A 109.73.179.95 A 87.221.209.204

Figure 12. Domain flux implementation.

Figures 13 illustrates the concept of FFSN; new IP addresses (109.117.185.235, 91.230.157.174) had been used instead of 185.1.62.82.

The figure displays four sequential packet captures from Wireshark, each showing a series of DNS queries and responses. The captures are titled '2015-07-08_capture-win8.pcap'. The first capture shows queries for 'top-web.org'. The second capture shows queries for 'serviceonline2013.org'. The third capture shows queries for 'servicewebcheck.org'. The fourth capture shows queries for 'serviceonline2013.org' and 'servicewebcheck.org'. Each packet entry includes columns for No., Time, Source, Destination, Protocol, Length, and Info.

Figure 13. FFSN implementation.

3. Packet capture of SSH connection tunneled in Iodine. Figure 14 shows the connection negotiation between the server and the client.

The figure shows a single packet capture titled 'Iodine.pcap'. It displays a long sequence of DNS traffic. The traffic starts with a query for 'nsl.canonical.com' and continues with numerous queries for '50A.local' and other domain names. The responses include standard query responses and 'Unknown operation (3) response' messages. The packet list table shows columns for No., Time, Source, Destination, Protocol, Length, and Info.

Figure 14. Iodine connection negotiation

By analyzing the traffic from the Wireshark capture, it was noticed that Iodine used some patterns or signatures in its negotiation between the server and the client. These patterns were appended to the subdomain (tunnel.ialabs.net), and included the following:

- aA-Aaahhh-Drink-mal-ein-j\344germeister-
 \x61\x41\x2d\x41\x61\x61\x68\x68\x68\x2d\x44\x72\x69\x6e\x6
 b\x2d\x6d\x61\x6c\x2d\x65\x69\x6e\x2d\x4a\xe4\x67\x65\x72\x
 6d\x65\x69\x73\x74\x65\x72\x2d\x06
- aA-La-fl\373te-na\357ve-fran\347aise-est-retir\351-\340-Cr\350te.
 \x61\x41\x2d\x4c\x61\x2d\x66\x6c\xfb\x74\x65\x2d\x6e\x61\xe
 f\x76\x65\x2d\x66\x72\x61\x6e\xe7\x61\x69\x73\x65\x2d\x65\x
 73\x74\x2d\x72\x65\x74\x69\x72\xe9\x2d\xe0\x2d\x43\x72\xe8\
 x74\x65\x06
- aAbBcCdDeEfFgGhHiIjJkKlLmMnNoOpPqQrRsStTuUvVwWxXyYzZ
 \x61\x41\x62\x42\x63\x43\x64\x44\x65\x45\x66\x46\x67\x47\x68\x48\
 x69\x49\x6a\x4a\x6b\x4b\x6c\x4c\x6d\x4d\x6e\x4e\x6f\x4f\x70\x50\x
 71\x51\x72\x52\x73\x53\x74\x54\x75\x55\x76\x56\x77\x57\x78\x58\x7
 9\x59\x7a\x5a\x06
- aA0123456789\274\275\276\277
 \x61\x41\x30\x31\x32\x33\x34\x35\x36\x37\x38\x39\xbc\xbd\xbe\xbf
- \300\301\302\303\304\305\306\307\310\311\312\313\314\315\316\317
- \320\321\322\323\324\325\326\327\330\331\332\333\334\335\336\337
- \340\341\342\343\344\345\346\347\350\351\352\353\354\355\356\357
- \360\361\362\363\364\365\366\367\370\371\372\373\374\375

The following figure shows the SSH connection tunneled in DNS packets. The SSH connection initialization started at packet No. 4248.

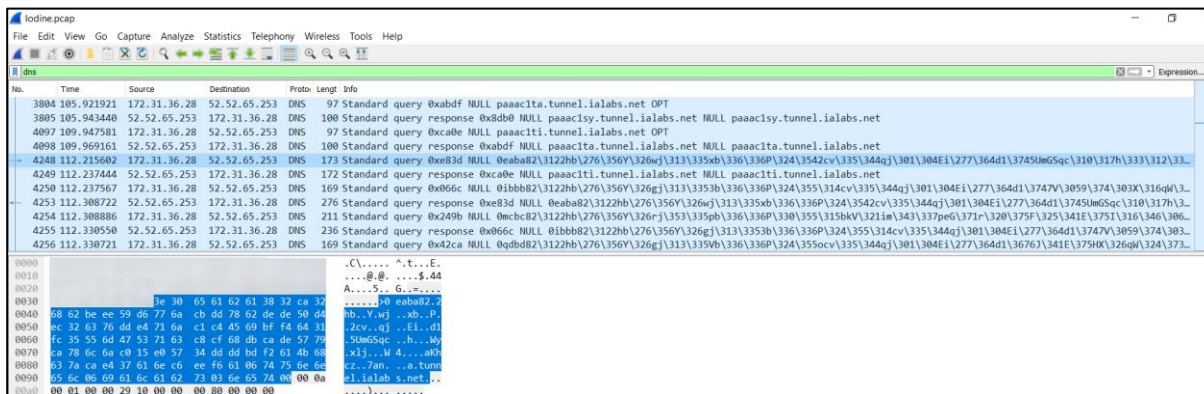


Figure 15. SSH connection tunneled in DNS packets (iodine).

Iodine uses the NULL RR (QTYPE 10) and provides higher performance by allowing the downstream data to be sent without encoding. Each DNS reply can contain over a kilobyte of compressed payload data. However, regarding the upstream data (the DNS requests), Iodine uses either Base-32 or a non-standard Base-64 encoding method (based on a configuration option) (Nussbaum, Neyron, & Richard, 2009).

By analyzing different SSH connections, the following signature was used to establish a SSH tunnel:

eaba82.2hb..Y.w which is equivalent to following hex string:

\x65\x61\x62\x61\x38\x32\xca\x32\x68\x62\xbe\xee\x59\xd6

\x77

4. **Packet capture of SSH connection tunneled in DNS2TCP.** The following figure shows the SSH connection tunneled in DNS packets. The SSH connection initialization started at packet No. 4248.

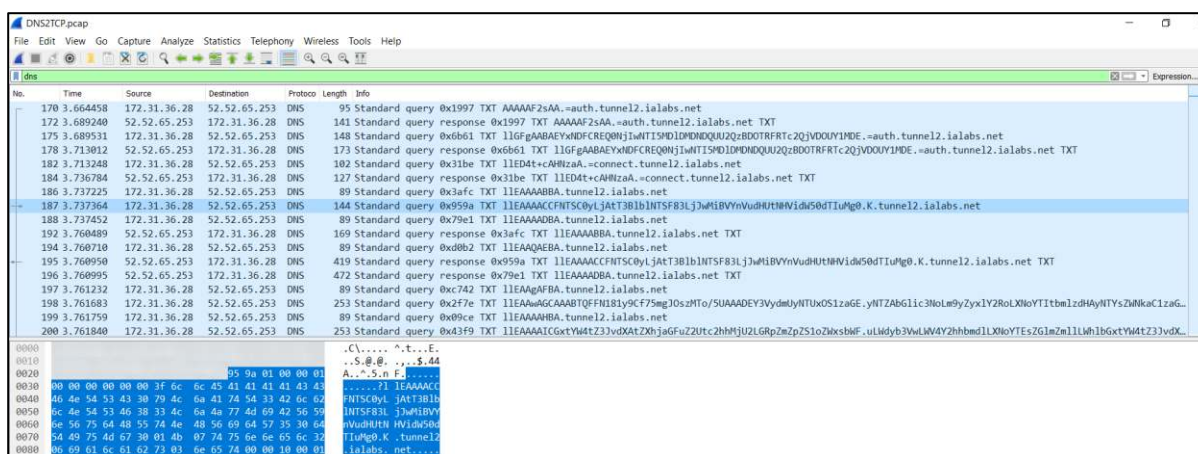


Figure 16. SSH connection tunneled in DNS packets (dns2tcp).

By default, DNS2TCP uses the TXT RR type. Since DNS2TCP uses Base-64 encoding, SSH connection packets can be analyzed to find the encoded pattern of the normal SSH connection. In a normal situation, a SSH connection contains the `SSH-2.0-OpenSSH_` string. By analyzing packet No. 187 in figure 16, the string `AAACCFNTSC0yLjAtT3BlblNTSF83LjJwMiBVYnVudHUtNHVidW50dTIuMg0` was used to establish a SSH tunnel. This string is equivalent to `SSH-2.0-OpenSSH_7.2p2 Ubuntu-4ubuntu2.2`.

To detect other versions of SSH-2.0 connections and/or operating systems, only the `CFNTSC0yLjAtT3BlblNTSF8` part is used to detect SSH connections tunneled in DNS packets. After decoding this Base-64 encoded string, it is equivalent

to `SSH-2.0-OpenSSH_`. To detect only `OpenSSH_` string, only the string `T3B1b1NTSF8` is used.

In BRO NSM, both Iodine and DNS2TCP queries were logged in DNS.Logs as lowercase queries, so the detection script was written based on the lowercase equivalents. For example, the string Iodine SSH tunnel `eaba82.2hb..y.w` was detected by the equivalent hex of its lowercase `eaba82.2hb..y.w`, which is `\x65\x61\x62\x61\x38\x32\xca\x32\x68\x62\xbe\xee\x79\xd6\x77` instead of `\x65\x61\x62\x61\x38\x32\xca\x32\x68\x62\xbe\xee\x59\xd6\x77`. To find the complete script for SSH tunneling detection, refer to Appendix D.

Summary

This chapter presented and analyzed the packet captures of DGA, FFSN, and DF implementations, as well as SSH connections tunneled in DNS tunneling using Iodine and DNS2TCP tools. The next chapter discusses the results, conclusion, and future work.

Chapter V: Results, Conclusion, and Recommendations

Introduction

As mentioned in the previous chapters, DNS protocol can be implemented for botnet communications. Some of these implementations utilize different evasion techniques to circumvent the detection and prevention methods. In this study, the multistage-detection technique was designed and implemented using the BRO Network Security Monitor. This chapter presents the results of the analysis methods, conclusion, and future recommendations.

Results

This thesis presents an empirical solution to detect DNS-based botnets at different stages of their communications. Domain Generation Algorithm-based botnets were detected by the percentage of unique NXDomain errors among the total DNS queries within an epoch of 30 minutes. The fluxing techniques—FFSN and DF—were detected by the percentage of total Server Failure errors among the total DNS queries within an epoch of two hours.

Also, this thesis presents some popular DNS tunneling tools that are used to tunnel botnet traffic in DNS packets, and presents a signature-based method to detect DNS-tunneled botnets that use SSH as their encryption algorithm. Each tool has different method to tunnel the traffic. Since SSH is utilized to encrypt the tunnel, it is recommended to look for SSH connections wrapped in the DNS packets by looking for the encoded patterns of the SSH connection requests. Some tools, such as

Iodine, encoded the traffic using a non-standard Base-64 encoding method and other tools like DNS2TCP using a Base-64 encoding method.

From the Iodine lab results, a non-standard encoded pattern for SSH connections was detected. In the DNS2TCP lab, a standard encoded pattern for SSH connections' handshaking was detected. These patterns can be used as signatures to detect SSH connection handshaking tunneled in Iodine and DNS2TCP tools.

The following questions and answers provide a summary of the proposed solutions:

Question 1: Was the proposed method able to detect DGA-based botnets?

Answer: Yes, based on unique NXDomain thresholds.

Question 2: Was the proposed method able to detect FFSN- and DF-based botnets?

Answer: Yes, based on Server Failure thresholds.

Question 3: Were the signatures able to detect SSH tunneling in the suggested tools?

Answer: Yes, the signatures were able to detect SSH tunneling in Iodine and DNS2TCP tools.

Organized and professional botmasters may develop other methods to bypass these detection mechanisms. The following questions and answers explain:

Question 1: Can the botmasters evade the detection of the DGA-, FFSN-, or DF-based botnets based on the NXDomain or Server Failure thresholds?

Answer: Yes, changing the frequency of the DNS requests will minimize the NXDomain errors within a specific epoch. Also, they may control the TTL values properly to avoid the high number of Server Failure responses.

Question 2: Can the botmasters evade the non-standard encoded signatures of the SSH connection handshaking?

Answer: Yes. Like other characteristics and parameters in the DNS traffic, they may change the non-standard encoding/decoding code.

Question 3: Can the botmasters evade the standard encoded signatures of the SSH connection handshaking?

Answer: Creating a large set of possible strings of the encoded SSH connection handshaking using the standard encoding methods provides a strong mechanism for SSH tunneling detection. For example, using Base-64, the encoded pattern of *ANSSH-2.0-OpenSSH_* is *QU5TU0gtMi4wLU9wZW5TU0hf*, but the encoded pattern of *ASSH-2.0-OpenSSH_* is *QVNTSC0yLjAtT3BibINTSF8=*, which is completely different, because Base-64 method takes every three bytes and encodes them into four bytes output, so the order of the *OpenSSH_* string within the packet gives different outputs.

However, creating a signatures list provides a strong mechanism for SSH connections tunneling detection; botmasters may change the trend to use non-standard encoding/decoding methods to tunnel these connections.

Conclusion

This thesis was to design and implement detection techniques for DNS-based botnets at different stages of communications. Using a BRO Network Security Monitor, the suggested solutions were able to detect the botnet traffic at the rallying stage—when finding the C&C server—as well as, detecting the SSH tunneling connections used to encrypt the traffic after finding the C&C server.

The detection of DGA-, FFSN-, and DF-based botnets was based on a threshold value of specific DNS response failures. Whereas, the detection of SSH tunneling was based on encoded patterns of SSH connection handshaking within two popular DNS tunneling tools. The suggested solutions were able to detect the SSH tunneling in Iodine and DNS2TCP. These tools can be utilized to tunnel the SSH connections in DNS-based botnets.

Future Work

In this thesis, tunneling SSH connections were implemented using two of the most popular DNS tunneling tools. In the future, analyzing other tools that support SSH connections, such as DNSCAT and OzymanDNS, and creating a large set of possible strings of the encoded SSH connection handshaking using the standard encoding methods will contribute to the detection efforts.

Regarding DGA and fluxing techniques, finding an automated way to set the thresholds of DNS queries, NXDomain, and Server Failure based on the behavior of the system will improve the detection mechanisms. The threshold settings need to be dynamically adjusted with the changing nature of communication which would

optimize false positives and negatives. Also, to minimize the false positive of fluxing detection, the DNS Server Failure error—resulting from fluxing implementations—needs to be differentiated from similar errors caused by other issues.

References

- Albors, J. (2017, January 12). *Fast Flux networks: What are they and how do they work?* Retrieved September 11, 2017, from <https://www.welivesecurity.com/2017/01/12/fast-flux-networks-work/>.
- Anderson, B., & Ekman, E. (2014). *Iodine*. Retrieved October 16, 2017, from <http://code.kryo.se/iodine/>
- Antonakakis, M., Perdisci, R., Nadji, Y., Vasiloglou, N., Abu-Nimeh, S., Lee, W., & Dagon, D. (2012). From throw-away traffic to bots: Detecting the rise of DGA-based malware. In *Proceedings of the 21st USENIX Conference on Security Symposium* (pp. 24-24). Bellevue, WA: USENIX Association.
- Bilge, L., Sen, S., Balzarotti, D., Kirda, E., & Kruegel, C. (2011). EXPOSURE: Finding malicious domains using passive DNS analysis. In *Proceedings of the 18th Annual Network and Distributed System Security Symposium, NDSS 2011*. San Diego, California, USA.
- Botnet*. (2017, March 23). Retrieved March 26, 2017, from <https://en.wikipedia.org/wiki/Botnet>.
- Bots and botnets—A growing threat*. (2016). Retrieved March 26, 2017, from <https://us.norton.com/botnet/>.
- Brandhorst, C. J., & Pras, A. (2006). DNS: A statistical analysis of name server traffic at local network-to-Internet connections. In C. Delgado Kloos, A. Marín, & D. Larrabeiti (Eds.), *EUNICE 2005-Proceedings of the 11th Open European Summer School and IFIP WG6.4/6.6/6.9 Workshop* (pp. 255-270). (IFIP

International Federation for Information Processing; Vol. 196). Berlin: Springer.

doi: 10.1007/0-387-31170-X_19

Bro Introduction. (n.d.). Retrieved March 26, 2017, from <https://www.bro.org/sphinx/intro/>.

Caglayan, A., Toothaker, M., Drapeau, D., Burke, D., & Eaton, G. (2009). Real-time detection of fast flux service networks. *Cybersecurity Applications & Technology Conference for Homeland Security* (pp. 285-292). Washington, DC: IEEE Computer Society. doi:10.1109/catch.2009.44.

Choi, H., Lee, H., & Kim, H. (2007). Botnet detection by monitoring group activities in DNS traffic. *CIT '07 Proceedings of the 7th IEEE International Conference on Computer and Information Technology* (pp. 715-720). Washington, DC: IEEE Computer Society.

Cantón, D. (2015, January 20). *Botnet detection through DNS-based approaches*. Retrieved September 11, 2017, from <https://www.certs.es/en/blog/botnet-detection-dns>.

CounterMeasures—A security blog. (2010, September 24). Retrieved March 26, 2017, from <http://countermeasures.trendmicro.eu/the-history-of-the-botnet-part-i/>.

Dabbagh, M. (2012). *Covert channels in botnets*. Retrieved from <http://web.engr.oregonstate.edu/~dabbaghm/projects/CovertChannels.pdf>.

Dietrich, C., Rossow, C., Freiling, F. C., Bos, H., Steen, M., & Pohlmann, N. (2011). On botnets that use DNS for command and control. *EC2ND '11 Proceedings of the 2011 Seventh European Conference on Computer Network Defense*

(pp. 9-16). Washington, DC: IEEE Computer Society.

doi:10.1109/EC2ND.2011.16

Domain name system. (2017, March 22). Retrieved March 27, 2017, from

https://en.wikipedia.org/wiki/Domain_Name_System.

Farnham, G., & Atlasis, A. (2013). *Detecting DNS tunneling*. SANS Institute InfoSec Reading Room (pp. 1-32).

Garcia, S. (2015). *Malware capture facility project*. Retrieved August 22, 2017, from

<https://stratosphereips.org>.

Hagen, J., & Luo, S. (2016, August 17). *Why Domain Generating Algorithms (DGAs)?*

Retrieved March 25, 2017, from <http://blog.trendmicro.com/domain-generatingalgorithms-dgas/>.

Holz, T., Gorecki, C., Rieck, K., & Freiling, F. (2008). Measuring and detecting fast-flux service networks. *In Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS)*.

HSC. (2012, May 02). Retrieved October 02, 2017, from <http://www.hsc.fr/ressources/outils/dns2tcp/>.

Jin, Y., Ichise, H., & Iida, K. (2015). Design of detecting botnet communication by monitoring direct outbound DNS queries. *CSCLOUD '15 Proceedings of the 2015 IEEE 2nd International Conference on Cyber Security and Cloud Computing (CSCloud)* (pp. 37-41). Washington, DC: IEEE Computer Society.
doi:10.1109/CSCloud.2015.53.

Krmíček, V. (2011). Inspecting DNS flow traffic for purposes of botnet detection.

GEANT3 JRA2 T4 Internal Deliverable (pp. 1-9).

Lysenko, S., Pomorova, O., Savenko, O., Kryshchuk, A., & Bobrovnikova, K. (2015).

DNS-based anti-evasion technique for botnets detection. *In Proceedings of the 8th IEEE International Conference on Intelligent Data Acquisition and*

Advanced Computing Systems: Technology and Applications (pp. 453-458).

Warsaw, Poland: IEEE Computer Society.

doi:10.1109/IDAACS.2015.7340777.

Newman, R. C. (2010). Computer security: Protecting digital resources. Sudbury, MA:

Jones and Bartlett.

Nussbaum, L., Neyron, P., & Richard, O. (2009). On robust covert channels inside

DNS. *Emerging Challenges for Security, Privacy and Trust IFIP Advances in Information and Communication Technology* (pp. 51-62). doi:10.1007/978-3-

642-01244-0_5.

Peer-to-peer botnets for beginners. (2013, December 22). Retrieved March 26, 2017,

from [https://www.malwaretech.com/2013/12/peer-to-peer-botnets-](https://www.malwaretech.com/2013/12/peer-to-peer-botnets-forbeginners.html)

[forbeginners.html](https://www.malwaretech.com/2013/12/peer-to-peer-botnets-forbeginners.html).

RFC 883-domain names-implementation and specification. (1983 November).

Retrieved March 26, 2017, from <https://www.ietf.org/rfc/rfc883.txt>.

RFC 4413-TCP/IP field behavior. (2006 March). Retrieved March 26, 2017, from

<https://tools.ietf.org/html/rfc4413>.

Rouse, M. (2012, February). *Botnet (zombie army)*. Retrieved March 26, 2017, from <http://searchsecurity.techtarget.com/definition/botnet>.

The Bro network security monitor. (n.d.). Retrieved March 26, 2017, from <https://www.bro.org/>.

What is a Botnet Attack?-Definition. (n.d.). Retrieved March 25, 2017, from <https://usa.kaspersky.com/internet-security-center/threats/botnet-attacks#.WN3sKYWcE2w>.

Yadav, S., & Reddy, A. L. (2012). Winning with DNS failures: Strategies for faster botnet detection. *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering Security and Privacy in Communication Networks* (pp. 446-459). doi:10.1007/978-3-642-31909-9_26

Zhang, H., Papadopoulos, C., & Massey, D. (2013). Detecting encrypted botnet traffic. *2013 Proceedings IEEE INFOCOM, 2013*, doi:10.1109/infcom.2013.6567180.

Zeltser, L. (2016, August 08). *Tunneling data and commands over DNS to bypass firewalls*. Retrieved March 27, 2017, from <https://zeltser.com/c2-dns-tunneling/>

Appendix A: DNA Message Format

Section 1 (Message Header)

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Message ID															
QR	OPCODE	AA	TC	RD	RA	res1	res2	res3	RCODE						
QDCOUNT (No. of items in Question Section)															
ANCOUNT (No. of items in Answer Section)															
NSCOUNT (No. of items in Authority Section)															
ARCOUNT (No. of items in Additional Section)															

Message ID 16-bit message ID supplied by the questioner and reflected back unchanged by the responder. Identifies the transaction.

QR Query - Response bit. Set to 0 by the questioner (query) and to 1 in the response (answer).

OPCODE Identifies the request/operation type. Currently assigned values are:

Value	Meaning/Use
0	QUERY. standard query.
1	IQUERY. Inverse query. Optional support by DNS
2	STATUS. DNS status request

AA Authoritative Answer. Valid in responses only. Because of aliases multiple owners may exist so the AA bit corresponds to the name which matches the query name, OR the first owner name in the answer section.

TC Truncation - specifies that this message was truncated due to length greater than that permitted on the transmission channel. Set on all truncated messages except the last one.

RD Recursion Desired - this bit may be set in a query and is copied into the response if recursion supported by this Name Server. If Recursion is rejected by this Name Server, for example it has been configured as Authoritative Only, the response (answer) does not have this bit set. Recursive query support is optional.

RA Recursion Available - this bit is valid in a response (answer) and denotes whether recursive query support is available (1) or not (0) in the name server.

RCODE Identifies the response type to the query. Ignored on a request (question).

Currently assigned values:

Value	Meaning/Use
0	No error condition.
1	Format error - The name server was unable to interpret the query.
2	Server Failure - The name server was unable to process this query due to a problem with the name server.
3	Name Error - Meaningful only for responses from an authoritative name server, this code signifies that the domain name referenced in the query does not exist.
4	Not Implemented - The name server does not support the requested kind of query.
5	Refused - The name server refuses to perform the specified operation for policy reasons. For example, a name server may

not wish to provide the information to the particular requester, or a name server may not wish to perform a particular operation (e.g., zone transfer) for particular data.

QDCOUNT Unsigned 16-bit integer specifying the number of entries in the Question Section.

ANCOUNT Unsigned 16-bit integer specifying the number of resource records in the Answer Section. May be 0 in which case no answer record is present in the message.

NSCOUNT Unsigned 16-bit integer specifying the number of name server resource records in the Authority Section. May be 0 in which case no authority record(s) is(are) present in the message.

ARCOUNT Unsigned 16-bit integer specifying the number of resource records in the Additional Section. May be 0 in which case no additional record(s) is(are) present in the message.

Section 2 (Question Section)

Field Name	Meaning/Use
QNAME	The domain name being queried
QTYPE	The resource records being requested
QCLASS	The Resource Record(s) class being requested, for instance, internet, chaos etc.

QNAME The name being queried, its content will depend upon the QTYPE (below), for example, a request for an A record will typically require a host part, such as, www.example.com, an MX query will only require a base domain name, such as, example.com. The name being queried is split into labels by

removing the separating dots. Each label is represented as a length/data pair as follows:

Value	Meaning/Use
no. of chars	Single octet defining the number of characters in the label which follows. The top two bits of this number must be 00 (indicates the label format is being used) which gives a maximum domain name length of 63 bytes (octets). A value of zero indicates the end of the name field.
domain name	A string containing the characters in the label.

QTYPE

Unsigned 16-bit value. The resource records being requested. These values are assigned by IANA and a complete list of values may be obtained from them. The following are the most commonly used values:

Value	Meaning/Use
x'0001 (1)	Requests the A record for the domain name
x'0002 (2)	Requests the NS record(s) for the domain name
x'0005 (5)	Requests the CNAME record(s) for the domain name
x'0006 (6)	Requests the SOA record(s) for the domain name
x'000B (11)	Requests the WKS record(s) for the domain name
x'000C (12)	Requests the PTR record(s) for the domain name
x'000F (15)	Requests the MX record(s) for the domain name
x'0021 (33)	Requests the SRV record(s) for the domain name
x'001C (28)	Requests the AAAA record(s) for the domain name

x'00FF (255) Requests ANY resource record (typically wants SOA, MX, NS and MX)

QCLASS Unsigned 16-bit value. The CLASS of resource records being requested e.g. Internet, CHAOS etc. These values are assigned by IANA and a complete list of values may be obtained from them. The following are the most commonly used values:

Value	Meaning/Use
x'0001 (1)	IN or Internet

Section 3 (Answer Section)

Field Name	Meaning/Use
NAME	The name being returned e.g. www or ns1.example.net If the name is in the same domain as the question then typically only the host part (label) is returned, if not then a FQDN is returned.
TYPE	The RR type, for example, SOA or AAAA
CLASS	The RR class, for instance, Internet, Chaos etc.
TTL	The TTL in seconds of the RR, say, 2800
RLENGTH	The length of RR specific data in octets, for example, 27
RDATA	The RR specific data (see Binary RR Formats below) whose length is defined by RLENGTH, for instance, 192.168.254.2

NAME This name reflects the QNAME of the question i.e. any may take one of TWO formats. The first format is the label format defined for QNAME above. The second format is a pointer. A pointer is an unsigned 16-bit value with the following format (the top two bits of 11 indicate the pointer format):

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
1	1	The offset in octets (bytes) from the start of the whole message. Must point to a label format record to derive name length.													

The offset in octets (bytes) from the start of the whole message. Must point to a label format record to derive name length.

Note: Pointers, if used, terminate names. The name field may consist of a label (or sequence of labels) terminated with a zero-length record OR a single pointer OR a label (or label sequence) terminated with a pointer.

TYPE Unsigned 16-bit value. The resource record types - determines the content of the RDATA field. These values are assigned by IANA and a complete list of values may be obtained from them. The following are the most commonly used values:

Value	Meaning/Use
x'0001 (1)	An A record for the domain name
x'0002 (2)	A NS record for the domain name
x'0005 (5)	A CNAME record for the domain name
x'0006 (6)	A SOA record for the domain name
x'000B (11)	A WKS record(s) for the domain name
x'000C (12)	A PTR record(s) for the domain name
x'000F (15)	A MX record for the domain name
x'0021 (33)	A SRV record(s) for the domain name
x'001C (28)	An AAAA record(s) for the domain name

CLASS Unsigned 16-bit value. The CLASS of resource records being requested, for example, Internet, CHAOS etc. These values are assigned by IANA and a complete list of values may be obtained from them. The following are the most commonly used values:

Value	Meaning/Use
x'0001 (1)	IN or Internet

TTL Unsigned 32-bit value. The time in seconds that the record may be cached.
A value of 0 indicates the record should not be cached.

RDLENGTH Unsigned 16-bit value that defines the length in bytes (octets) of the RDATA record.

RDATA Each (or rather most) resource record types have a specific RDATA format which reflect their resource record format as defined below:

SOA

Value	Meaning/Use
Primary NS	Variable length. The name of the Primary Master for the domain. May be a label, pointer or any combination.
Admin MB	Variable length. The administrator's mailbox. May be a label, pointer or any combination.
Serial Number	Unsigned 32-bit integer.
Refresh interval	Unsigned 32-bit integer.
Retry Interval	Unsigned 32-bit integer.
Expiration Limit	Unsigned 32-bit integer.
Minimum TTL	Unsigned 32-bit integer.

MX

Value	Meaning/Use
Preference	Unsigned 16-bit integer.
Mail Exchanger	The name host name that provides the service. May be a label, pointer or any combination.

A

Value	Meaning/Use
IP Address	Unsigned 32-bit value representing the IP address

AAAA

Value	Meaning/Use
IP Address	16 octets representing the IP address

PTR, NS

Value	Meaning/Use
Name	The host name that represents the supplied IP address (in the case of a PTR) or the NS name for the supplied domain (in the case of NS). May be a label, pointer or any combination.

Section 4 (Authority Section)

The Resource Record(s) which point to the domain authority. These authority records have exactly the same format as Section 3 (Answer Section) it is simply their position in an authority section that determines they are authority records (and that they will be of TYPE NS).

Section 5 (Additional Section)

The Resource Record(s) which may hold additional information. These additional records have exactly the same format as Section 3 (Answer Section) it is simply their position in an additional section that determines they are additional records.

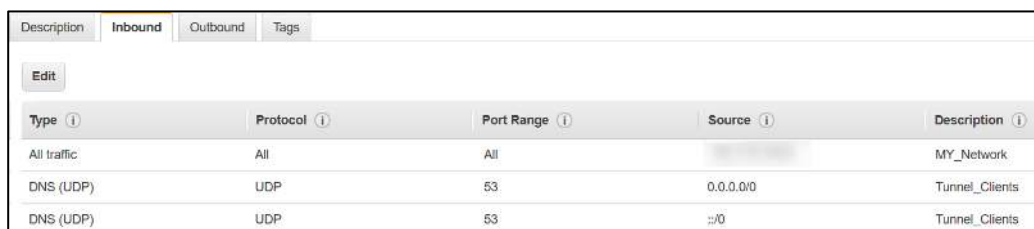
Appendix B: Iodine Lab Implementation

- 1- Create two Ubuntu instances on Amazon Web Services.
- 2- Assign static IP addresses to these instances.
- 3- Install Iodine on both instances

```
sudo apt-get install iodine
```

- 4- Configure the inbound traffic on the server to allow only the incoming DNS traffic.

Note: For experiment purposes, all inbound traffic was allowed to the server and to the client only from my remote location.

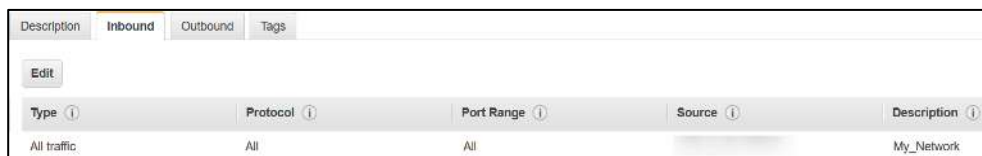


Type	Protocol	Port Range	Source	Description
All traffic	All	All		MY_Network
DNS (UDP)	UDP	53	0.0.0.0/0	Tunnel_Clients
DNS (UDP)	UDP	53	:::0	Tunnel_Clients

Figure 17. DNS tunneling server firewall (inbound configuration).

- 5- Configure the inbound traffic on the client to block all incoming traffic.

Note: For experiment purposes, all inbound traffic was allowed to the client only from my remote location.



Type	Protocol	Port Range	Source	Description
All traffic	All	All		My_Network

Figure 18. DNS tunneling client firewall (inbound configuration).

- 6- Block the outbound SSH traffic on the client.



Type	Protocol	Port Range	Destination	Description
DNS (UDP)	UDP	53	0.0.0.0/0	Allow Only DNS

Figure 19. DNS tunneling client firewall (outbound configuration).

- 7- Register a domain name (ialabs.net) and delegate a subdomain (tunnel.ialabs.net) to the DNS tunneling server.

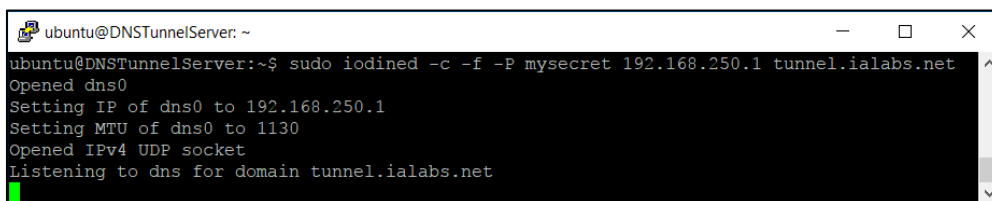
A	tunnelns	52.52.65.253
NS	tunnel	tunnelns.ialabs.net

Figure 20. Domain name settings.

- 8- Using Putty, connect to the Iodine server at 52.52.65.253.

- 9- Run iodined on the tunneling server

```
sudo iodined -c -f -P mysecret 192.168.250.1
tunnel.ialabs.net
```



```
ubuntu@DNSTunnelServer: ~
ubuntu@DNSTunnelServer:~$ sudo iodined -c -f -P mysecret 192.168.250.1 tunnel.ialabs.net
Opened dns0
Setting IP of dns0 to 192.168.250.1
Setting MTU of dns0 to 1130
Opened IPv4 UDP socket
Listening to dns for domain tunnel.ialabs.net
```

Figure 21. Running iodined on the DNS tunneling server.

```

ubuntu@DNSTunnelServer: ~
ubuntu@DNSTunnelServer:~$ ifconfig
dns0    Link encap:UNSPEC  HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
        inet addr:192.168.250.1  P-t-P:192.168.250.1  Mask:255.255.255.224
        UP POINTOPOINT RUNNING NOARP MULTICAST  MTU:1130  Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:500
        RX bytes:0 (0.0 B)  TX bytes:0 (0.0 B)

eth0    Link encap:Ethernet  HWaddr 06:58:56:ee:c6:1c
        inet addr:172.31.1.197  Bcast:172.31.15.255  Mask:255.255.240.0
        inet6 addr: fe80::458:56ff:feee:c61c/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST  MTU:9001  Metric:1
        RX packets:1971487 errors:0 dropped:0 overruns:0 frame:0
        TX packets:2093724 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:312388290 (312.3 MB)  TX bytes:5106522083 (5.1 GB)

lo      Link encap:Local Loopback
        inet addr:127.0.0.1  Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING  MTU:65536  Metric:1
        RX packets:13595955 errors:0 dropped:0 overruns:0 frame:0
        TX packets:13595955 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1
        RX bytes:52564537614 (52.5 GB)  TX bytes:52564537614 (52.5 GB)

```

Figure 22. Tunneling interface on the server (192.168.250.1).

- 10- Make sure that tunnel is setup and working [troubleshooting your iodine setup](#).

```

Troubleshoot your iodine setup

Analyzing DNS setup for tunnel domain 'tunnel.ialabs.net'... (might take some time)

Looking for nameserver for ialabs.net.. got ns09.domaincontrol.com (at 216.69.185.5).
Resolving delegation of tunnel.ialabs.net at 216.69.185.5... to tunnelns.ialabs.net (at 52.52.65.253).

Expecting iodined to be accessible at 52.52.65.253... yes, using proto 00000502.
Testing iodine reply using default nameserver... ok.

Well done, your iodine setup seems fine!

Try again
Back to iodine

```

Figure 23. DNS tunneling server troubleshooting.

- 11- Using Putty, connect to the Iodine client at 35.165.67.21.
- 12- Run iodine on the tunneling client

```
sudo iodine -f -P mysecret 52.52.65.253 tunnel.ialabs.net
```

```

ubuntu@DNSTunnelClient: ~
ubuntu@DNSTunnelClient:~$ sudo iodine -f -P mysecret 52.52.65.253 tunnel.ialabs.
net
Opened dns0
Opened IPv4 UDP socket
Sending DNS queries for tunnel.ialabs.net to 52.52.65.253
Autodetecting DNS query type (use -T to override).
Using DNS type NULL queries
Version ok, both using protocol v 0x00000502. You are user #0
Setting IP of dns0 to 192.168.250.2
Setting MTU of dns0 to 1130
Server tunnel IP is 192.168.250.1
Testing raw UDP data to the server (skip with -r)
Server is at 172.31.1.197, trying raw login: ...failed
Using EDNS0 extension
Switching upstream to codec Base128
Server switched upstream to codec Base128
No alternative downstream codec available, using default (Raw)
Switching to lazy mode for low-latency
Server switched to lazy mode
Autoprobing max downstream fragment size... (skip with -m fragsize)
768 ok.. 1152 ok.. ...1344 not ok.. ...1248 not ok.. ...1200 not ok.. 1176 ok..
1188 ok.. will use 1188-2=1186
Setting downstream fragment size to max 1186...
Connection setup complete, transmitting data.

```

Figure 24. Running iodine on the DNS tunneling client.

```

ubuntu@DNSTunnelClient: ~
ubuntu@DNSTunnelClient:~$ ifconfig
dns0    Link encap:UNSPEC HWaddr 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00
        inet addr:192.168.250.2 P-t-P:192.168.250.2 Mask:255.255.255.224
        UP POINTOPOINT RUNNING NOARP MULTICAST MTU:1130 Metric:1
        RX packets:0 errors:0 dropped:0 overruns:0 frame:0
        TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:500
        RX bytes:0 (0.0 B) TX bytes:0 (0.0 B)

eth0    Link encap:Ethernet HWaddr 06:00:5e:0c:74:a6
        inet addr:172.31.36.28 Bcast:172.31.47.255 Mask:255.255.240.0
        inet6 addr: fe80::400:5eff:fe0c:74a6/64 Scope:Link
        UP BROADCAST RUNNING MULTICAST MTU:9001 Metric:1
        RX packets:1034310 errors:0 dropped:0 overruns:0 frame:0
        TX packets:1220462 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1000
        RX bytes:207458770 (207.4 MB) TX bytes:3091990273 (3.0 GB)

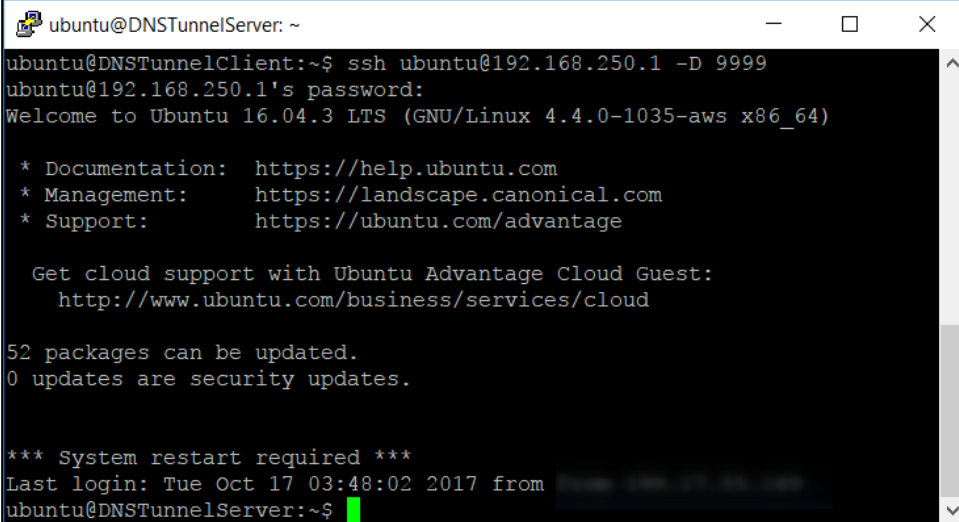
lo      Link encap:Local Loopback
        inet addr:127.0.0.1 Mask:255.0.0.0
        inet6 addr: ::1/128 Scope:Host
        UP LOOPBACK RUNNING MTU:65536 Metric:1
        RX packets:16516615 errors:0 dropped:0 overruns:0 frame:0
        TX packets:16516615 errors:0 dropped:0 overruns:0 carrier:0
        collisions:0 txqueuelen:1
        RX bytes:26991675594 (26.9 GB) TX bytes:26991675594 (26.9 GB)

```

Figure 25. Tunneling interface on the server (192.168.250.2).

13- Run the following command

```
ssh ubuntu@192.168.250.1 -D 9999
```


A terminal window titled 'ubuntu@DNSTunnelServer: ~' showing an SSH connection. The user runs 'ssh ubuntu@192.168.250.1 -D 9999'. The terminal displays the password prompt, a welcome message for Ubuntu 16.04.3 LTS, system information, and update notifications. It also shows a system restart requirement and the last login time.

```
ubuntu@DNSTunnelServer: ~  
ubuntu@DNSTunnelClient:~$ ssh ubuntu@192.168.250.1 -D 9999  
ubuntu@192.168.250.1's password:  
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-1035-aws x86_64)  
  
* Documentation:  https://help.ubuntu.com  
* Management:    https://landscape.canonical.com  
* Support:       https://ubuntu.com/advantage  
  
Get cloud support with Ubuntu Advantage Cloud Guest:  
http://www.ubuntu.com/business/services/cloud  
  
52 packages can be updated.  
0 updates are security updates.  
  
*** System restart required ***  
Last login: Tue Oct 17 03:48:02 2017 from [REDACTED]  
ubuntu@DNSTunnelServer:~$
```

Figure 26. SSH connection tunneling (iodine).

Appendix C: DNS2TCP Lab Implementation

- 1- Create two Ubuntu instances on Amazon Web Services.
- 2- Assign static IP addresses to these instances.
- 3- Install dns2tcp on both instances by running the following command:

```
sudo apt-get install dns2tcp
```

- 4- Configure the inbound traffic on the server to allow only the incoming DNS traffic.

Note: For experiment purposes, all inbound traffic was allowed to the server and to the client only from my remote location.

Type ⓘ	Protocol ⓘ	Port Range ⓘ	Source ⓘ	Description ⓘ
All traffic	All	All	0.0.0.0/0	MY_Network
DNS (UDP)	UDP	53	0.0.0.0/0	Tunnel_Clients
DNS (UDP)	UDP	53	:::0	Tunnel_Clients

Figure 27. DNS tunneling server firewall (inbound configuration).

- 5- Configure the inbound traffic on the client to block all incoming DNS traffic.

Note: For experiment purposes, all inbound traffic was allowed to the client only from my remote location.

Description Inbound Outbound Tags				
Edit				
Type ⓘ	Protocol ⓘ	Port Range ⓘ	Source ⓘ	Description ⓘ
All traffic	All	All	0.0.0.0/0	My_Network

Figure 28. DNS tunneling client firewall (inbound configuration).

- 6- Block the outbound SSH traffic on the client.

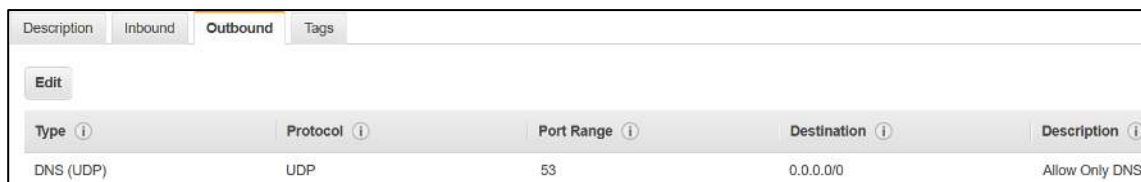


Figure 29. DNS tunneling client firewall (outbound configuration).

- 7- Register a domain name (ialabs.net) and delegate a subdomain (tunnel2.ialabs.net) to the DNS tunneling server.

A	tunnelns	52.52.65.253
NS	tunnel2	tunnelns.ialabs.net

Figure 30. Domain name settings.

- 8- Using Putty, connect to the dns2tcp server at 52.52.65.253.
 9- Configure the `dns2tcpd.conf` as the following:

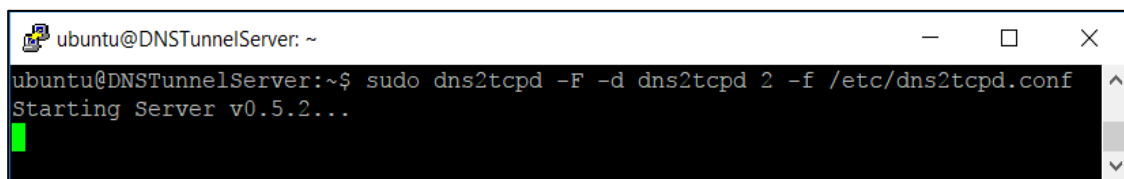
```
listen = 0.0.0.0
port = 53
## If you change this value, also change the USER variable
in /etc/default/dns2t$
user = ubuntu          ## tunneling server user name
chroot = /usr/local/src/
pid_file = /var/run/dns2tcp.pid
key = secretkey
domain = tunnel2.ialabs.net  ## tunneling domain
resources = ssh:127.0.0.1:22
```

- 10- Using Putty, connect to the dns2tcp client 35.165.67.21.
 11- Configure the `dns2tcpc.conf` as the following:

```
domain = tunnel2.ialabs.net    ## tunneling domain
resource = ssh
local_port = 8888
key = secretkey
debug_level = 3
server = 216.69.185.5 ## IP address of the name server
```

- 12- Run the dns2tcp server using the following command on the tunneling server:

```
sudo dns2tcpd -F -d dns2tcpd 2 -f /etc/dns2tcpd.conf
```

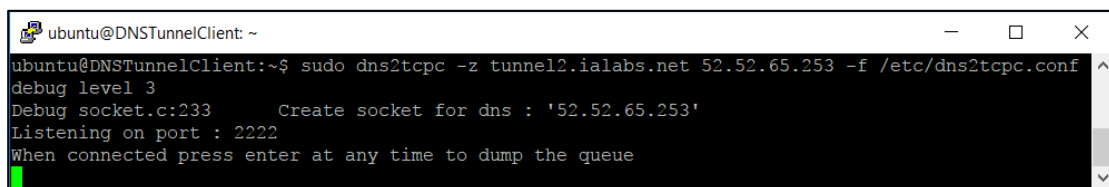
A terminal window titled 'ubuntu@DNSTunnelServer: ~' showing the command 'sudo dns2tcpd -F -d dns2tcpd 2 -f /etc/dns2tcpd.conf' being executed. The output is 'Starting Server v0.5.2...' followed by a green cursor.

```
ubuntu@DNSTunnelServer: ~
ubuntu@DNSTunnelServer:~$ sudo dns2tcpd -F -d dns2tcpd 2 -f /etc/dns2tcpd.conf
Starting Server v0.5.2...
█
```

Figure 31. Running dns2tcp on the DNS tunneling server.

- 13- Run the dns2tcp client using the following command on the tunneling client:

```
sudo dns2tcpd -z tunnel2.ialabs.net 52.52.65.253 -f
/etc/dns2tcpd.conf
```

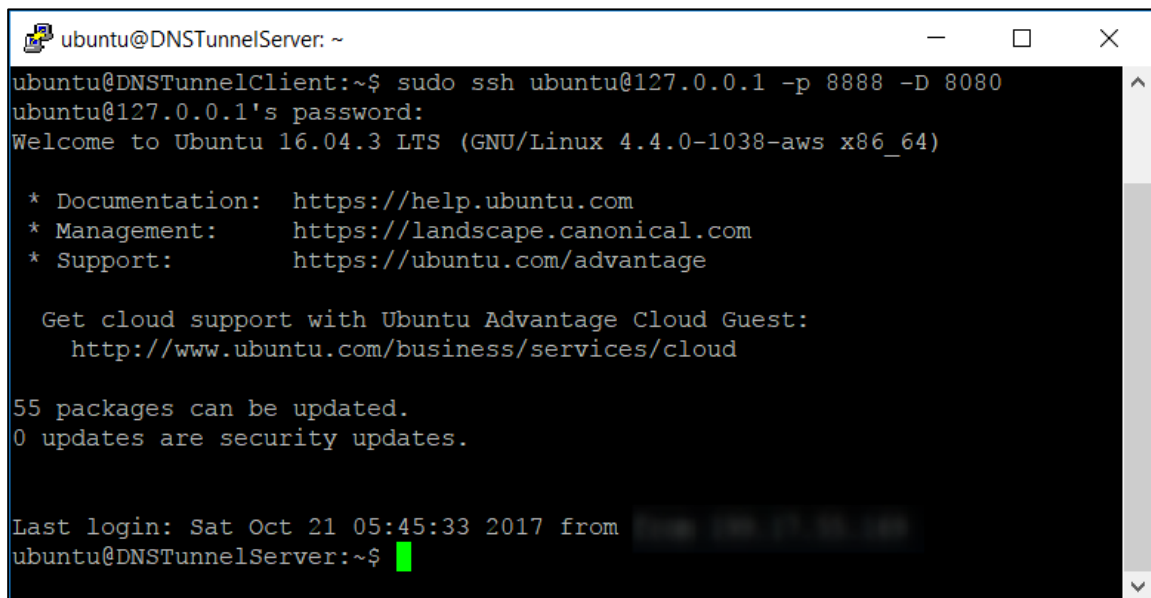
A terminal window titled 'ubuntu@DNSTunnelClient: ~' showing the command 'sudo dns2tcpd -z tunnel2.ialabs.net 52.52.65.253 -f /etc/dns2tcpd.conf' being executed. The output includes 'debug level 3', 'Debug socket.c:233 Create socket for dns : '52.52.65.253'', 'Listening on port : 2222', and 'When connected press enter at any time to dump the queue' followed by a green cursor.

```
ubuntu@DNSTunnelClient: ~
ubuntu@DNSTunnelClient:~$ sudo dns2tcpd -z tunnel2.ialabs.net 52.52.65.253 -f /etc/dns2tcpd.conf
debug level 3
Debug socket.c:233 Create socket for dns : '52.52.65.253'
Listening on port : 2222
When connected press enter at any time to dump the queue
█
```

Figure 32. Running dns2tcp on the DNS tunneling client.

- 14- Run the following command on the tunneling client:

```
sudo ssh ubuntu@127.0.0.1 -p 8888 -D 8080
```

A terminal window titled 'ubuntu@DNSTunnelServer: ~' with standard window controls. The terminal shows a user on 'ubuntu@DNSTunnelClient' running the command 'sudo ssh ubuntu@127.0.0.1 -p 8888 -D 8080'. The output shows a successful SSH connection to 'ubuntu@127.0.0.1's password:' followed by a Ubuntu 16.04.3 LTS login banner. The banner includes links for documentation, management, and support, as well as information about cloud support and package updates. The session ends with the user logging out and returning to the 'ubuntu@DNSTunnelServer' prompt.

```
ubuntu@DNSTunnelClient:~$ sudo ssh ubuntu@127.0.0.1 -p 8888 -D 8080
ubuntu@127.0.0.1's password:
Welcome to Ubuntu 16.04.3 LTS (GNU/Linux 4.4.0-1038-aws x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:    https://landscape.canonical.com
 * Support:       https://ubuntu.com/advantage

Get cloud support with Ubuntu Advantage Cloud Guest:
  http://www.ubuntu.com/business/services/cloud

55 packages can be updated.
0 updates are security updates.

Last login: Sat Oct 21 05:45:33 2017 from [REDACTED]
ubuntu@DNSTunnelServer:~$
```

Figure 33. SSH connection tunneling (dns2tcp).

Appendix D: BRO Network Security Monitor Scripts

DGA Detection Script

```

event dns_request(c: connection, msg: dns_msg, query: string, qtype: count, qclass: count)
{
    if ( c$id$resp_p == 53/udp && query != "" )
        SumStats::observe("dns.lookup", [$host=c$id$orig_h], [$str=query]);
}

event dns_query_reply(c: connection, msg: dns_msg, query: string, qtype: count, qclass: count)
{
    if (c$dns?$rcode_name && c$dns$rcode == 3)
        SumStats::observe("DGA", [$host=c$id$orig_h], [$str=query]);
}

const time_interval = 30 mins &redef;
const dns_threshold = 50 &redef;
const min_dns = 10 &redef;

global w = 1.00 &redef;

event bro_init()
{
    local q1 = SumStats::Reducer($stream="dns.lookup", $apply=set(SumStats::SUM));
    SumStats::create([$name="DNS Query",
        $epoch=time_interval,
        $reducers=set(q1),
        $epoch_result(ts: time, q_key: SumStats::Key, q_result: SumStats::Result) =
        {
            local q = q_result["dns.lookup"];
            print "##### TOTAL DNS QUERIES #####";
            print fmt("%s SENT A TOTAL OF %d DNS QUERIES WITHIN 30 MINUTES", q_key$host, q$num);
            w = q$sum;
        },
        $epoch_finished(ts: time) =
        {
            print "";
        }
    ]]);

    local x1 = SumStats::Reducer($stream="DGA", $apply=set(SumStats::UNIQUE));
    SumStats::create([$name="NXDomain",
        $epoch=time_interval,
        $reducers=set(x1),
        $epoch_result(ts1: time, x_key: SumStats::Key, x_result: SumStats::Result) =
        {
            local x = x_result["DGA"];
            print "***** TOTAL NXDOMAINS *****";
            print fmt("%s RECEIVED A TOTAL OF %d NXDOMAIN RESPONSES - %d WERE UNIQUE &
REPRESENTED %.0f%% OF THE TOTAL DNS REQUESTS", x_key$host, x$num, x$unique, ((x$unique+0.0)/(w+0.0)*100));
            if ((w >= dns_threshold && x$unique/w >= 0.08) || (w < dns_threshold && w >=
min_dns && x$unique/w >= 0.40))
                print fmt ("%s MAY HAVE A RUNNING DGA!", x_key$host);
            print
            "*****";
            print "";
        },
        $epoch_finished(ts1: time) =
        {
            print "";
        }
    ]]);
}

```

```
Terminal - wasseem@wj-vm: /opt/bro/share/bro/policy/misc/699
File Edit View Terminal Tabs Help

##### TOTAL DNS QUERIES #####
10.0.2.15 SENT A TOTAL OF 845 DNS QUERIES WITHIN 30 MINUTES

***** TOTAL NXDOMAINS *****
10.0.2.15 RECEIVED A TOTAL OF 837 NXDOMAIN RESPONSES - 837 WERE UNIQUE & REPRESENTED 99% OF THE TOTAL DNS REQUESTS
10.0.2.15 MAY HAVE A RUNNING DGA!
*****

##### TOTAL DNS QUERIES #####
10.0.2.15 SENT A TOTAL OF 967 DNS QUERIES WITHIN 30 MINUTES

***** TOTAL NXDOMAINS *****
10.0.2.15 RECEIVED A TOTAL OF 956 NXDOMAIN RESPONSES - 956 WERE UNIQUE & REPRESENTED 99% OF THE TOTAL DNS REQUESTS
10.0.2.15 MAY HAVE A RUNNING DGA!
*****

##### TOTAL DNS QUERIES #####
10.0.2.15 SENT A TOTAL OF 648 DNS QUERIES WITHIN 30 MINUTES

***** TOTAL NXDOMAINS *****
10.0.2.15 RECEIVED A TOTAL OF 641 NXDOMAIN RESPONSES - 605 WERE UNIQUE & REPRESENTED 93% OF THE TOTAL DNS REQUESTS
10.0.2.15 MAY HAVE A RUNNING DGA!
*****
```

Figure 34. DGA detection—script output.

FLUX Detection Script

```

event dns_request(c: connection, msg: dns_msg, query: string, qtype: count, qclass: count)
{
    if ( c$cid$resp_p == 53/udp && query != "" )
        SumStats::observe("dns.lookup", [ $host=c$cid$orig_h ], [ $str=query ]);
}

event dns_rejected(c: connection, msg: dns_msg, query: string, qtype: count, qclass: count)
{
    if ( c$dns?$rcode_name && c$dns?$rcode == 2 )
        SumStats::observe("Flux", [ $host=c$cid$orig_h ], [ $str=query ]);
}

const time_interval = 30 mins &redef;
const dns_threshold = 50 &redef;
const min_dns = 8 &redef;

global w = 1.00 &redef;

event bro_init()
{
    local q1 = SumStats::Reducer($stream="dns.lookup", $apply=set(SumStats::SUM));
    SumStats::create([ $name="DNS Query",
        $epoch=time_interval,
        $reducers=set(q1),
        $epoch_result(ts: time, q_key: SumStats::Key, q_result: SumStats::Result) =
        {
            local q = q_result["dns.lookup"];
            print "##### TOTAL DNS QUERIES #####";
            print fmt("%s SENT A TOTAL OF %d DNS QUERIES WITHIN 30 MINUTES", q_key$host, q$num);
            w = q$sum;
        },
        $epoch_finished(ts: time) =
        {
            print "";
        }
    ]);

    local f1 = SumStats::Reducer($stream="Flux", $apply=set(SumStats::SUM));
    SumStats::create([ $name="SERVFAIL",
        $epoch=time_interval,
        $reducers=set(f1),
        $epoch_result(ts: time, f_key: SumStats::Key, f_result: SumStats::Result) =
        {
            local f = f_result["Flux"];
            print "***** TOTAL SERVFAIL *****";
            print fmt("%s RECEIVED A TOTAL OF %d SERVER FAILURE RESPONSES WHICH REPRESENTED
%.0f%% OF THE TOTAL DNS REQUESTS", f_key$host, f$num, ((f$sum+0.00)/(w+0.00)*100));
            if ((w >= dns_threshold && f$sum/w >= 0.06) || (w < dns_threshold && w >= min_dns
&& f$sum/w >= 0.26))
                print fmt ("%s MAY HAVE A FLUX IMPLEMENTATION!", f_key$host);
            print
            "*****";
            print "";
        },
        $epoch_finished(ts1: time) =
        {
            print "";
        }
    ]);
}

```



```
Terminal - wasseem@wj-vm: /opt/bro/share/bro/policy/misc/699
File Edit View Terminal Tabs Help

##### TOTAL DNS QUERIES #####
10.0.2.108 SENT A TOTAL OF 8 DNS QUERIES WITHIN 30 MINUTES

***** TOTAL SERVFAIL *****
10.0.2.108 RECEIVED A TOTAL OF 6 SERVER FAILURE RESPONSES WHICH REPRESENTED 75% OF THE TOTAL DNS REQUESTS
10.0.2.108 MAY HAVE A FLUX IMPLEMENTATION!
*****

##### TOTAL DNS QUERIES #####
10.0.2.108 SENT A TOTAL OF 1 DNS QUERIES WITHIN 30 MINUTES

##### TOTAL DNS QUERIES #####
10.0.2.108 SENT A TOTAL OF 1 DNS QUERIES WITHIN 30 MINUTES

##### TOTAL DNS QUERIES #####
10.0.2.108 SENT A TOTAL OF 4 DNS QUERIES WITHIN 30 MINUTES

***** TOTAL SERVFAIL *****
10.0.2.108 RECEIVED A TOTAL OF 3 SERVER FAILURE RESPONSES WHICH REPRESENTED 75% OF THE TOTAL DNS REQUESTS
*****

##### TOTAL DNS QUERIES #####
10.0.2.108 SENT A TOTAL OF 5 DNS QUERIES WITHIN 30 MINUTES

***** TOTAL SERVFAIL *****
10.0.2.108 RECEIVED A TOTAL OF 5 SERVER FAILURE RESPONSES WHICH REPRESENTED 100% OF THE TOTAL DNS REQUESTS
*****
```

Figure 35. Flux detection—script output

SSH Tunneling Detection Script

```

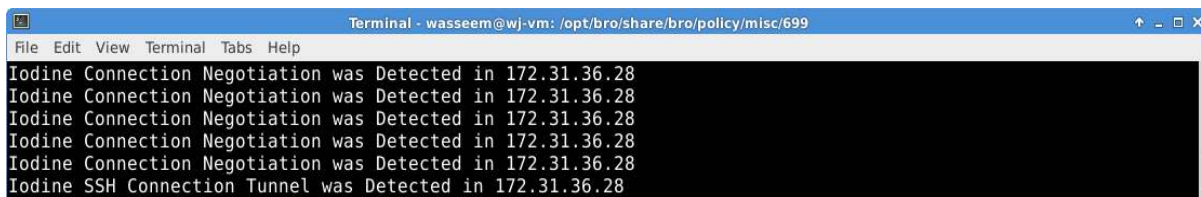
event dns_query_reply(c: connection, msg: dns_msg, query: string, qtype: count, qclass: count)
{
    if ( c$Id$resp_p == 53/udp && query != "" )

        if ("aabbccddeeffgghhiijjkkllmmnnnooppqrrssttuuvvwwxxyyzz" in query
            || "aa-aaahhh-drink-mal-ein-j\344germeister-." in query
            || "aa-la-fl\373te-na\357ve-fran\347aise-est-retir\351-\340-cr\350te." in query
            || "aa0123456789\274\275\276\277" in query
            || "aa\300\301\302\303\304\305\306\307\310\311\312\313\314\315\316\317" in query
            || "aa\320\321\322\323\324\325\326\327\330\331\332\333\334\335\336\337" in query
            || "aa\340\341\342\343\344\345\346\347\350\351\352\353\354\355\356\357" in query
            || "aa\360\361\362\363\364\365\366\367\370\371\372\373\374\375" in query)
            print fmt ("Iodine Connection Negotiation was Detected in %s", c$Id$orig_h);

        # eaba82.2hb..Y.w Detection
        if ("\x65\x61\x62\x61\x38\x32\xca\x32\x68\x62\xbe\xee\x79\xd6\x77" in query)
            print fmt ("Iodine SSH Connection Tunnel was Detected in %s", c$Id$orig_h);

        # CFNTSC0yLjAtT3BlblNTSF8 Detection
        if ("\x63\x66\x6e\x74\x73\x63\x30\x79\x6c\x6a\x61\x74\x74\x33\x62\x6c\x62\x6c\x6e\x74\x73\x66\x38"
            in query)
            print fmt ("DNS2TCP SSH Connection Tunnel was Detected in %s", c$Id$orig_h);
    }
}

```



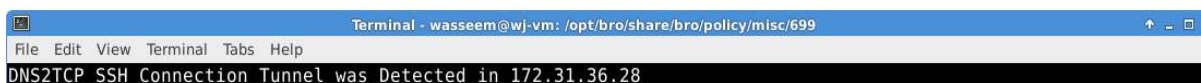
A terminal window titled "Terminal - wasseem@wj-vm: /opt/bro/share/bro/policy/misc/699" displays the following output:

```

Iodine Connection Negotiation was Detected in 172.31.36.28
Iodine Connection Negotiation was Detected in 172.31.36.28
Iodine Connection Negotiation was Detected in 172.31.36.28
Iodine Connection Negotiation was Detected in 172.31.36.28
Iodine Connection Negotiation was Detected in 172.31.36.28
Iodine SSH Connection Tunnel was Detected in 172.31.36.28

```

Figure 36. Iodine SSH tunneling detection.



A terminal window titled "Terminal - wasseem@wj-vm: /opt/bro/share/bro/policy/misc/699" displays the following output:

```

DNS2TCP SSH Connection Tunnel was Detected in 172.31.36.28

```

Figure 37. DNS2TCP SSH tunneling detection.

Appendix E: DGA-, FFSN-, and DF-based Botnets Dataset

DGA-based botnet dataset download link

https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-7/2013-08-20_capture-win1.pcap

FFSN and DF-based botnet dataset download link

https://mcfp.felk.cvut.cz/publicDatasets/CTU-Malware-Capture-Botnet-127-2/2015-07-08_capture-win8.pcap