Titan: A Tool for Safe and Faithful Capture of Network Fingerprints of a Bot

By
Osama Haq
NUST201260797MSEECS61312F

Supervisor
Dr. Muhammad Usman Ilyas
Department of Computing

A thesis submitted in partial fulfillment of the requirements for the degree of Masters of Science in Computer Science (MS CS)

In
School of Electrical Engineering and Computer Science,
National University of Sciences and Technology (NUST),
Islamabad, Pakistan.

(April 2015)

Approval

It is certified that the contents and form of the thesis entitled "**Titan: A Tool for Safe and Faithful Capture of Network Fingerprints of a Bot** " submitted by **Osama Haq** have been found satisfactory for the requirement of the degree.

Advisor: **Dr. Muhammad Usman Ilyas**

Signature: ____________________

Date: ____________________

Committee Member 1: **Dr. Affan A. Syed**

Signature: ____________________
Date: ____________________

Committee Member 2: **Dr. Ali Khayam**

Signature: ____________________
Date: ____________________

Committee Member 3: **Dr. Aamir Shafi**

Signature: ____________________
Date: ____________________

Dedication

I dedicate this thesis to my family, friends and colleagues that inspired and motivated me throughout this degree.

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: **Osama Haq**

Signature: ________________

Acknowledgment

I would like to thank my Guidance and Evaluation Committee for their support throughout this thesis. It would have not been possible without their encouragement. I would also like to thank all the SysNet lab members who have helped me during this journey.

Table of Contents

List of Tables

List of Figures

Abstract

Botnets are an evolutionary form of malware, unique in requiring network connectivity, for herding by a botmaster, that allows coordinated attacks as well as dynamic evasion from detection. Thus, the most interesting features of a bot relate to its rapidly evolving network behavior.

The few academic and commercial malware observation systems that exist, however, do not safely and faithfully capture fingerprints of a bot. Moreover, these systems are either proprietary or have large cost and management overhead. We observe that the network behavior of bots is largely dependent upon the containment policy and changes considerably under different operational contexts.

We first propose an iterative and semi automated way to contain harmful activity generated by bots and then identify these various contexts that can impact its fingerprint. We also present Titan: a system that generates faithful network fingerprints by recreating all these contexts and stressing the bot with different network settings and host interactions. This effort includes a semi-automated and tunable containment policy to prevent bot proliferation. Most importantly, Titan has low cost overhead as a minimal setup requires just two machines, while the provision of a user-friendly web interface reduces the setup and management overhead.

We then show a fingerprint of Kanav F bot to demonstrate the bootstrap capturing feature of Titan. We also show a fingerprint of the Cryptolocker bot to demonstrate automatic detection of its domain generation algorithm (DGA) and its evolution over the period of six months. Finally, we demonstrate the effective identification of context-specific behavior with a controlled deployment of Zeus botnet.

Chapter 1

Introduction

Botnets are networks of malware infected machines, called *bots*, which are under the control of an external entity called *botmaster*. A botmaster uses command and control (C&C) server to provide configurational as well as evolutionary updates to the infected machines.

Botnets are different from traditional malware as they are primarily used for financial gains. A botmaster can rent a subset of their bots for carrying out large scale distributed attacks (such as DDoS or SPAM) as well as for information stealing (such as credit card numbers, online banking username and passwords) [1]. Recent studies have shown existence of large underground economy based on botnets [2–4]. Thus, to gain the maximum market share (in terms of number of machines), black-hat hackers continue to innovate their infection and evasion techniques.

Security companies and researchers analyze malware by studying their binaries (computer program that is used to infect machines). There are two types of analysis performed on a malicious binary, host based analysis and network based analysis. In host based analysis, the effects of binary on the host's operating system are analyzed whereas in network based analysis, the network traffic generated by the infected machine is analyzed. The resulting output of host and network based analysis systems is used by security companies to improve their malware detection and removal solutions.

1.1 Motivation

Botnets have emerged as one of the most prominent cyber security threats over the last decade. Today, botnets are used by criminal entities for stealing personal and financial information, launching cyber attacks and sending out spam emails. The most defining feature of a bot, that separates it from other classes of malware like viruses, trojans and worms, is its communication with the botmaster. A master imparts his instructions to the bot infected machines using a command and control (C&C) server. After a bot infects a machine, its main purpose is to lie dormant and act on any instructions from its C&C server. Although the output of host based analysis can be used to eradicate a bot infection from a machine, network based analysis of botnet is required to understand the behavior of bot and for identification of its C&C servers.

There are many tools that can be used for network analysis of botnet binaries, some are proposed in academia [5–7] and some of them are available on the web [8, 9]. Security companies also perform analysis using their proprietary systems and regularly publish reports on threats [10, 11]. We observe that there are three major shortcomings in the way network analysis of botnets is currently being performed.

Firstly, when a bot's network activity is analyzed, the issue of containing harmful network activity arises. If a bot is allowed to freely communicate with external entities, it might launch an attack (like DoS, Spam) and spread to other systems in its local network. If it is not allowed to communicate with outside world, its true behavior cannot be captured. Thus, rendering the analysis useless and resulting fingerprints incomplete. We observe that most of the publicly available analysis systems as well as solutions proposed in academia do not tackle this issue and choose to ignore it altogether [12]. The tools which do take in to account the issue of containment require manual analysis of traffic logs and generation of containment policies requires a human resource all the time [5, 6], hence, proving costly. The security companies who analyze botnet binaries also do not disclose their containment methodology and policies to the user. We believe that containment is an integral part of network analysis and should be incorporated in any analysis tool for effective and safe fingerprint generation.

Secondly, since a behavior of a bot depends on various environmental conditions like network configuration, location, hardware and software of infected machine, these analysis systems do not take these factors into account when performing analysis. A bot regularly sends reports to its C&C server, which contain information about any user activity along with hardware and software of the infected machine. Based on these reports sent by the bot, a botmaster may choose to assign different roles to infected machines [13]. These roles are heavily dependent on the environment of the infected machine. A machine with a public IP instead of a NATed IP might start acting a stepping stone to other bot infected machines, similarly a machine with higher computing resources might be used for bitcoin mining. A botmaster may also detect if a bot is being analyzed in virtual environment and cease any network activity. Thus, behavior of a bot is dependent on the instructions received from botmaster and may change from time to time. We, however, observe that botnet network analysis tools and systems, do not take into account this changing behavior of bots when performing analysis. Hence, the output fingerprints of these systems are partial and do not truly identify behavior of a bot. We believe that in order to fully understand botnets, these environmental conditions should be identified and a bot must be rigorously analyzed in these conditions.

Thirdly, current botnet analysis systems are either proprietary or have high setup and management cost [5, 6, 12]. Furthermore along with limited output, very little information is available for anyone who wants to replicate and enhance the system. Most of the tools that are proposed in academia require high performance machines and an elaborate network architecture. Although, some tools available as web based services require simpler network architectures but lack analysis in different environmental contexts and do not have effective containment policies. Hence providing incomplete network fingerprints to the user.

While most of the existing work focuses on providing great details about host based behavior of bots, there is very little focus on providing detailed network fingerprints [14] [12]. While we can only speculate about the reason for such restricted disclosure, we strongly feel the need for a botnet fingerprinting system that is open source ,can be set up with minimal cost and management overhead. This systems should also provide detailed network behavior of a bot under various operational contexts and effective containment policies. Such a system will be a very useful tool for the academic research community, especially new entrants, to deeply study and understand the threat and then build and evolve their defenses.

In this thesis we will present Titan, a low overhead botnet analysis tool. Titan recreates, using SDN and user emulation techniques, various operational contexts necessary for eliciting network behavior of the bot. Furthermore, Titan has a semi-automated containment engine that minimizes manual tuning of policies to contain any harmful effects of a bot (attacks or proliferation). It also provides a hierarchical and multifaceted view of the network fingerprints generated, which allows intermediate users to quickly identify salient features of the bot, while allowing advanced users to drill down into greater detail. Titan is open source, easy to setup and provides extensibility to developers due to its modular design.

1.2 Contribution

In this thesis, we set out to design and implement various features of Titan: a botnet analysis tool. Titan is an open source, low cost, easy to setup and manage botnet analysis tool that examines bot binaries under various environmental settings, provides semi automation of containment policies and generates network fingerprints in a hierarchical and user friendly format.

In accordance with our problem statement, We now lay out our research objectives and the corresponding contributions of this thesis.

- **Objective 1:** Incorporating safety measures for effective fingerprint generation.

  Contribution: The aim of our objective is to provide a method for effective fingerprint generation using automated containment policies. We first present our methodology for containment and then address design challenges in incorporating containment features in Titan. We discuss our SDN based implementation of containment policies in great detail and provide a way to minimize human intervention in generation of containment policies. We show the efficacy of our containment policies by analyzing real world bots under various policies.

- **Objective 2:** Identification and implementation of operational contexts that impact a bot's behavior.

  Contribution: We first identify various host and network based contexts that affect behavior of bots on a victim machine. These operational contexts include network environment, type of infected machine, user activity on the machine and observation duration of analysis. The aim of identifying these contexts is to faithfully capture fingerprints of a bot. These contexts help us in design of our botnet analysis tool: Titan. We present our design challenges and decisions in incorporating these operational contexts in Titan and provide implementation details for each context. We validate effective fingerprint generation by deploying a custom botnet infrastructure and analyzing the bot in different network configurations.

- **Objective 3:** Minimizing cost, setup and usage overheads in botnet analysis tools.

  Contribution: We present challenges faced in building a low cost systems that can be setup with minimal deployment and management overhead. We present our user friendly fingerprint and discuss its generation. We provide our methodology for generating fingerprints from deluge of logs generated by Titan. We analyze two variants of cryptolocker botnet and show how our fingerprint features help us identifying its evolution over time.

Chapter 2

Background and Literature Review

2.1 Background

There are two types of analysis performed on botnet executables, host based analysis and network based analysis. In host based analysis, the effects of a bot binary on the host operating systems is analyzed whereas in network based analysis, the network communication carried out by the bot binary is analyzed. Majority of the botnet analysis systems perform host based analysis and there are very few systems that indulge in network analysis of botnet binaries.

The analysis systems performing network analysis of bots do not cater its changing nature due to its connectivity with botmaster. A bot may change its behavior in order to fully utilize the available resources on the infected machine. This change in behavior depends on multiple factors, we call each of these factors operational contexts.

Based on activity a bot performs on an infected machine, fingerprints are generated which consist of a summary/highlights of these activities. This information is useful to researchers studying botnet behaviors as well as to botnet detection systems. However, a bot may be involved in malicious activities and may spread to other systems within the network or launch an attack against any one on the Internet. This problem is known as the issue of containment. In the coming section, we will present notable work in the field of network analysis of botnets.

2.2 Literature Review

Much work has been carried out on host based analysis of botnets [15–21], however there have been very few systems that specifically perform network analysis of botnets [5–7, 13]. In this section, we present a summary of major work carried out in network fingerprinting of botnets as they form the vast majority of NIDS-based botnet detection systems [5–7].. Our thesis presents a low cost, easy to setup and manage botnet analysis tool with multiple operational contexts, automated containment policies, and effective output fingerprints. We further divide our related work into these goals identified above and review existing tools that are proposed in academia along with online analysis services. We also focus on many online services [8, 14, 22] that perform both host based and network based analysis of malware.

2.2.1 Containment Policies

With the rise of botnets over the past few years, the issue of containing harmful traffic when analyzing bots has also gained some prominence. Most of the analysis systems have some sort of static containment policies in place [7, 23], however only GQ [5] and botlab [6] discuss formation of containment policies as a part of analysis. We now describe these systems and their containment policies.

Our inspiration for this work mainly came from GQ [5]. In GQ, Kreibich et al present the problem of containment in detail. They develop an architecture for malware execution, describe a manual approach for development of containment policies and also present their operational experiences in developing different containment policies. The main goal of GQ is to make containment development a natural step in malware analysis. Their approach towards containment is deny all out going traffic and iteratively allowing understood activity by analyzing all the blocked traffic manually. Although GQ highlights a lot of issues regarding containment but never fully explains the actual containment policies developed for stopping different type of known attacks. The malware analysis approach presented requires manual formation of containment policies for each analysis.

Botlab [6] also highlights the issue of containment in malware analysis. Their approach is to block all traffic destined towards privileged and vulnerable ports. They also enforce limits on data transmitted and no of outgoing going connections along with redirecting all the spam traffic towards a spam hole. All of the outgoing traffic is also monitored by a human operator as well. Although botlab had effective containment policies, it was decided by John et al that unknowingly allowing network attacks posses too much risk and the network fingerprinting aspect of botlab was shutdown.

Rajab et al containment policy during their study of botnets was to prevent engagement in outbound attacks [7]. They block outbound traffic on popular ports (e.g.135, 139, 445) and rate-limit each outgoing connection. They also detect IRC connections on application level and allow only those IRC communication.

Although we implement some static containment policies like those in Botlab and Rajab IRC, we also have dynamic containment policies. These policies are based on observed attacks and unlike GQ, they are developed in an automated fashion.

2.2.2 Capturing bot behavior in multiple operational contexts

Botnet analysis tools proposed rarely focus on capturing bot behavior in multiple operational contexts. While objective of these systems is to gain insight into botnets, their analysis does not take into account changing botnet behavior in various scenarios like different network settings, user activity on infected machine and analysis on multiple operating systems. However, there are some systems that do take into account execution environment of a bot binary as bots are known to hide their behavior in virtual analysis environment. We now discuss the systems that try to capture to behavior in multiple operational contexts.

GQ tries to capture bot behavior in both virtual and physical execution environment. GQ uses a network of virtual machines and raw iron machines, called inmates, for bot execution. GQ's architecture comprises of various components like containment server, logging engine, packet router, NATing module and vlan learning bridge. Moreover, GQ has the capability to control each traffic flow generating from a bot infected machine. It can also execute multiple bots on different machines at the same time.

Although GQ has all the necessary hardware and software to execute a bot in different contexts like multiple operating systems, network configurations and high performance bot execution machines. It only tries to capture bot behavior in different execution environment.

John et al use botlab setup to study the behavior of spamming botnets. They extract malicious URLs from incoming spam feed of university of Washington. Malware binaries are downloaded from those urls and executed in virtual as well as in baremetal environment. The resulting fingerprint from both executions is compared to deduce bot changing behavior in virtual environment. However, the focus of their study is to find trends in incoming and outgoing spam.

Our system captures bot behavior in different environmental settings including operating systems, execution environments, network settings and user activity. None of the related work in this area covers these operational contexts.

2.2.3 Fingerprint Generation

The network fingerprints generated by analysis tools are used by botnet detection systems like BotHunter [24], BotMiner [25], FireEye [26], and Damballa [27]. These detection systems uses this fingerprint information to enhance their signature database and to strengthen their correlation modules. We now present some of the existing systems that generate network fingerprints by analyzing malware binaries.

Abu Rajab et al botnet analysis system generates two types of network fingerprints, fnet and firc. The fnet fingerprint of a bot represents the network features of a bot whereas the firc fingerprint represents the IRC features of bot. Both of these fingerprints are generated in the execution phase of the analysis, where all the network traffic generated by a bot is dumped and analyzed. The fnet portion of fingerprint is a collection of DNS, IP,Port and scan behaviour of bot. The firc fingerprint comprises of IRC related feature of bot which are PASS,NICK,USER,MOD,JOIN. The two fingerprints although limited provide a way to join a botnet in the wild. The IRC feature of the bot is collected by application level pattern matching of traffic generated by bot, hence the system also captures botnets that operate on non-standard ports. This fingerprint forms the basis for custom IRC clients called drones that authors launch in order to study botnets as a whole.

In order to create a behavioral signature of a bot, botlab generates a network fingerprint. It does so by executing a binary in a sandbox environment and logging all the network connection attempts made by the bot. Botlab regards set of network flows as the fingerprint of the bot. This flow information includes protocol, IP address, DNS address and port of the server contacted by downloaded binary. Botlab also executes the binary twice in order to eliminate random connections generated by the binary. In order to detect different variants of a bot binary, john et al also compare the fingerprints of different executables and calculate the coefficient of similarity. In addition to the detection of polymorphism, this fingerprint information is also used to classify spam bot on the basis sent emails.

The network fingerprint defined by Rossow et al in Sandnet is similar to that of botlab. Their definition of fingerprint includes layer 4 protocol, source IP addr, destination IP addr, source port, destination port of all IPs contacted by the bot. Instead of deploying popular approach of using a sandbox for traffic analysis, Sandnet lets the malware talk to its C&C server under certain containment policies.

Online malware analysis services [8, 14, 22] do not address the issue of containment, and present no details on network traffic is handled in their systems. These systems although provide both host and network features of a bot, their main focus is on host based analysis. The network based features that these services provide are limited to any outgoing communication observed in five minutes of analysis.

Although there are tons of host based analysis systems proposes in the academia and available as online service that provide OS fingerprint of a malware in great detail but include very limited information about network fingerprint of a bot binary. Our system tries provide detailed information about network related fingerprints of bots, these include command and control server, network attack and behavior features.

2.2.4 Cost, setup, and management of botnet analysis systems

There are two type of cost, setup, and management approaches followed developers when in botnet analysis systems. The first approach is to make the analysis systems easier with the help of virtual machines on simple, commodity hardware [23]. The second approach is to setup a complex network architecture in order to collect and analyze malware [5–7]. The first approach although easier to manage and setup does not grant researchers fine control over the over their analysis experiment and environment whereas the second approach gives that level of control to researchers. We now discuss the cost, management and setup overheads in notable analysis systems.

Botnet analysis systems proposed by christian et al , john et al and abu rajab et al require multiple machines to setup [5–7]. In order to fully function, these machines need to be powerful in terms of their processor and RAM speed as well as storage. These systems also have complex network architecture that require expert networking and systems knowledge to setup and manage.

Sandnet [23] by Rossow et al and several online services [12] [9] [28] follow a different approach for malware analysis. Their analysis systems consist of 1-4 machines that use popular virtual machine software to deploy their analysis systems. There are no specific hardware requirements for these host systems and managing and setup cost is very low as compared to other systems proposed.

Our system has an easy to implement architecture that utilizes cheap and commodity hardware. We do however require network and system administration knowledge.

Chapter 3

Titan: Design and Architecture

3.1 Design Challenges

We now outline design challenges in building a low-cost and user-friendly system that *faithfully* captures the network features of a bot. These challenges stem from two basic questions: what constitutes a faithful network fingerprint for a bot, and what impacts the cost and overhead of existing systems with similar objectives?

3.1.1 Faithful Bot Fingerprints

We expect to capture all important network characteristics of a bot. We define a compact representation of *all* important characteristics as a *"faithful"* fingerprint.

Defining Faithful Features

Previous work has identified some important network characteristics of particular bots, (such as C&C type [29–31] and attack vectors [30, 31]), and also pointed out how bot behavior changes under different network and user stimuli [31, 32]. However, we have not encountered a system that can systematically generate bot fingerprints for all variations of these stimuli. We believe that only a system that can elicit all these behaviors can claim to generate a faithful fingerprint. Thus our first challenge is to identify and replicate the various operational contexts that can impact a bot's network behavior.
Network configuration: Bot binaries can behave differently based on different network configurations. We know that certain bots act as stepping stones in the bot obfuscation architecture when they have public ac-

cess [33]. Alternatively, the same binaries when in a NAT-ed environment show markedly different network characteristics.
Type of machine: Some bot binaries tend to be silent when deployed on a virtual machine [6], under the assumption that they are being monitored. Thus the type of infected machine also greatly impacts the network behavior of a bot.
User Activity: Malware responsible for information stealing and click hijacking triggers this behavior only when a user visits specific locations, such as banking or social network websites [31]. Thus, without some selective user interaction, this crucial aspect of a bot fingerprint cannot be obtained.

Factors Affecting Faithfulness of Fingerprint

Observation duration: Rossow et al. point out that the behavior of a bot immediately after infection (bootstrapping phase[1]) is markedly different from its post-bootstrapping phase [23]. We thus need to provide a tunable observation duration such that a faithful fingerprint is generated considering both phases of a bot lifecycle.
Containment: Any fingerprint generation system needs a containment policy that restricts attack and malicious activity for technical (our IPs will be blacklisted) and ethical reasons [5]. However, care has to be taken in not making this policy too restrictive, as otherwise the "in-the-wild" behavior of a bot cannot be understood.

3.1.2 Cost, Setup, and Management Overheads

A second set of challenges arises in building a low-cost system, that can be set up with minimal deployment and management overhead. Our philosophy is that anyone with intermediate network administration knowledge should be able to rapidly deploy and generate fingerprints.

The cost and setup overhead of the existing malware-analysis systems arises from requiring several high performance machines and an elaborate network architecture [5, 6]. Similarly, manual tweaking of containment policies (like in GQ [5]) entails the cost of employing a qualified professional, while simultaneously creating a time bottleneck in generating fingerprints. A key challenge here is thus to automate, as much as possible, containment of attack traffic without compromising on capturing other aspects of a bot's behavior. We believe that fully automated containment is not possible due to

[1]Thus a bot can try contacting its C&C server for configuration or updates, as well as detecting its network conditions.

the evolvable nature of botnets; a key challenge thus is to minimize manual intervention through *semi-automated* generation of a containment policy.

3.1.3 Incorporating Flexibility and Usability

A final challenge is to provide a user friendly representation of the network fingerprint. Such a system will naturally generate a deluge of logs and network events. We therefore want to provide fingerprints in a format that makes high-level features immediately obvious, yet still makes it possible to drill down and get finer granularity of information if desired. This feature will greatly enhance the usability of this tool, thus aiding security researchers in understanding and defending against the scourge of botnets.

3.2 Titan: Architecture

Our main focus in designing Titan is to ensure the generation of faithful bot fingerprints while minimizing cost and setup overheads. We begin by describing its architectural components and subsequently provide an overview of the fingerprinting workflow.

Figure 3.1 shows the main architectural components of Titan, each of which we briefly describe here.

3.2.1 Web Front End

The web frontend allows a user to not only submit the binary for evaluation, but also easily configure parameters that impact the fingerprint generation. The user is shown the possible operational contexts in which the binary can be executed. There are three context families, each with two operational contexts listed below:

- **network configuration context** of ① public or ② private
- **execution machine context** over ③ Virtual or ④ bare-metal machine
- **user activity context** in emulating access to ⑤ banking or ⑥ social media sites.

The binary is by default tested in each of these contexts, but users may deselect one or more of them. Advanced users can additionally vary the duration of observation for each experiment.

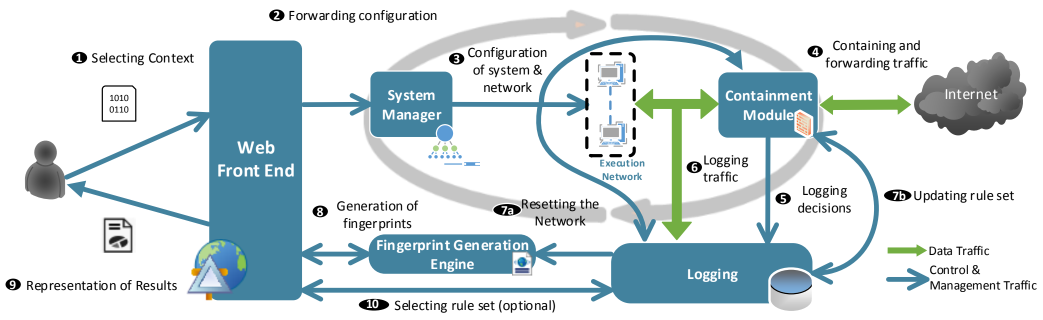

❷ Forwarding configuration
❶ Selecting Context
1010 0110
❸ Configuration of system & network
❹ Containing and forwarding traffic
Web Front End
System Manager
Execution Network
Containment Module
Internet
❻ Logging traffic
❺ Logging decisions
7b Updating rule set
❽ Generation of fingerprints
7a Resetting the Network
Fingerprint Generation Engine
Logging
Data Traffic
Control & Management Traffic
❾ Representation of Results
10 Selecting rule set (optional)

Figure 3.1: Titan Architecture and Workflow: From seeding of a bot binary to a faithful fingerprint generation

3.2.2 System Manager

The system manager is the glue that holds the entire system together. It chooses (and in the VM case, spawns) the machine to infect with the provided binary.

It takes other configuration parameters and converts them into commands to create the appropriate operational context for bot execution. Finally, in order to facilitate the automation of a containment policy (details below), the system manager is also responsible for stopping and restarting a new execution iteration.

3.2.3 Execution Network

The execution network is a setup of host machines (virtual and/or physical) that carry out execution of bots in different settings. This network contains, besides the infected machine, additional machines for logging and services emulation (like Dionaea [34]), as well as passive companion machines to emulate a populated LAN environment. The network configuration is set by the system manager according to the experiment requirements.

3.2.4 Containment and Logging Modules

The containment module is responsible for automating, as much as possible, the traffic filtration rules to prevent unwanted consequences of monitoring a bot. This containment policy is generated in an iterative manner. We start with a very conservative policy of blocking most traffic from the bot. After each iteration, we scan the traffic generated by the bot, continuously logged as flows by the containment module, to white-list connections that are purely C&C related and not dangerous. We then restart another iteration with an updated containment policy, that allows greater communication for the bot. We default to three iterations for every operational context (network setting or user behavior) in which the binary is executed. After which, users are provided an option to manually update the containment policy using their own heuristics to determine the safe connections.

3.2.5 Fingerprint Generation Engine

We populate an XML fingerprint schema for each operational context, specifying important characteristics therein. Moreover, we compare these XML files to further identify whether, and how, the network behavior changes between different operational contexts. This fingerprint is then provided to the web front which displays it in a format where a user is easily able to drill down for the detail of any important characteristic.

3.3 Titan: Workflow

Figure 3.1 shows the steps in a typical experiment on Titan. First, the user uploads a bot binary and selects appropriate operational contexts (described previously) from the web application interface (Step ❶). These settings along with the bot binary are sent to the system manager (Step ❷) which is responsible for the configuration and management of the execution network. The system manager then spawns VMs or initializes baremetal machines and also creates the appropriate network environment. The execution machine downloads the bot binary and any user emulation scripts from the network manager and execute them locally (Step ❸). In Step ❸ the system manager also initializes the containment and logging modules. The containment module forwards or blocks any traffic generated from the execution machine and logs the appropriate containment decisions (Step ❹ and ❺). The traffic generated to and from execution machines is also logged continuously by the logging module (Step ❻). Once a single iteration is over, the system manager resets the execution network and reinfects the machine (Step ❼a); meanwhile, the containment policy is updated with a more liberal list to allow greater C&C communication (Step ❼b). We explain the formation and revision of this rule set in Section 4.1.

The fingerprint generation engine applies heuristics on the logged data, to extract important network features of the bot, once all iterations for every operational context selected ends[2] (Step ❽). These fingerprints are then passed on to the web application, which presents them in a compact but efficient format for the user (Step ❾).

Advanced users are also presented with the final containment rule set which they manually tune and then rerun the bot for additional iterations for more detailed analysis (Step ❿). This step is the only one requiring manual intervention after the initial bot binary is provided to Titan, and is the reason we call our containment policy "semi-automated"; this too, however, is an optional step and intended only for advanced researchers.

[2]For 3 contexts families having two members each, this defaults to 18 iterations.

Chapter 4

Titan: Implementation and Evaluation

4.1 Implementation Details

We now describe the current implementation of Titan. We utilize just *two* desktop-class machines, a Gateway (4GB, 2.6Ghz) and a VM execution platform (3GB, 2.9 GHz) — in line with our desire for a low cost deployment that any research lab can put together (Figure 4.1). The Gateway implements all but one of the architectural modules for Titan. The execution network is emulated on the VM execution platform, connected to the gateway (providing NAT service) through a physical L2 switch, using a Xen hypervisor [35] that can spawn virtual machines (WinXP/Win7) and create context-specific network configurations. A third machine is connected via the switch, to act as a physical environment; we have a PXE boot system that uses Trinity Rescue Kit (TRK) [36] to allow experiments on a baremetal machine.

Titan supports analyzing a single bot at a time. This choice was consciously made as allowing parallel bot analysis (like in GQ [5]) excessively complicates our system to the detriment of our low-overhead objective.

We now describe in detail how we implement the individual components of Titan.

4.1.1 System Manager

The system manager is the brains of our system; it is responsible for creating the different operational contexts as well as for the iterative execution of a complete experiment. We first describe how we create different contexts and then divulge the details of iterative execution.

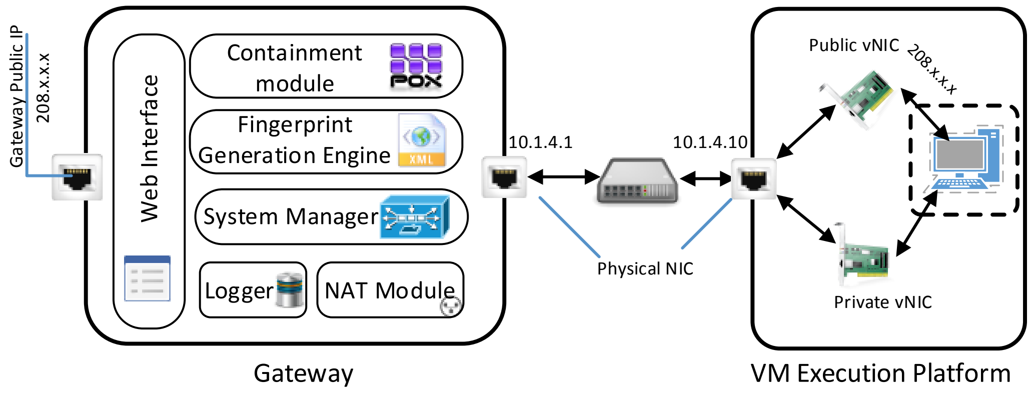


Figure 4.1: Titan Implementation using just two machines: A Gateway and a Virtual Execution Platform.

Execution and Operating System Context

In the execution machine context, a bot binary is tested on both virtual machine as well as on a physical machine.

We use Xen for our VM execution platform, where we clone a clean, base-image VM for infection. This image has a Windows operating system (XP or 7) with two virtual NICs. These NICs are used to create the network contexts where one of them is removed by system manager to achieve a particular public or NAT-ed environment. Some features of this base-image that can generate network traffic like system and application updates are disabled. This helps us in identifying traffic generated by bot by reducing white-traffic. The base-image also has DNS cache disabled, to accurately capture the communication patterns of our infected machine. Currently, it takes 5 minutes to clone a base-image and infect it with a malicious binary (Figure 4.1). For bare-metal execution, we employ the network (PXE) boot option to first load the Trinity Rescue Kit (TRK) [36] into the RAM of our baremetal machine. When executing this context, we first change the DHCP configuration of Gateway machine to allow PXE boot based connections and temporarily disable Gateway to allow TRK migration to the execution machine.

With TRK loaded, the baremetal machines acts like a client to listen for connections to transfer a disk image file. We place a clean base image of the corresponding Windows image on the Gateway machine where the *mclone* utility (essentially the TRK server) is initialized to transfer the selected image [1]. The system manager disables the PXE boot option from DHCP after the image transfer allowing the machine to now boot into its new operating system. Currently, the image size is around 4GB and it takes around 5 minutes for a successful cloning operation over a 100Mbps LAN connection.

User Activity Context

The user activity context consists of emulating different types of Internet activity on the infected machine. These activities include visiting social media sites or popular banking websites and entering login credentials. Currently, our system tries to emulate these two activities and see if any *extra* activity occurs to indicate the exfiltration of these credentials.

The user activity scripts are written in a windows automation language called AutoIt [37]. Initially, these scripts are placed on the Gateway along with the bot binary. Once a VM or physical machine successfully boots and acquires proper network configuration, the base-image is pre-configured to use sftp to transfer these scripts, along with the bot binary, from a defined location on the Gateway. First, the bot binary is executed on the analysis machine then, depending upon the selected context, the AutoIt scripts are executed. Currently these scripts visit social networking sites (facebook and twitter) and post as fake (but configurable) accounts, and also attempt logging in to banking and merchant sites (BoA, Wells Fargo, Amazon).

Our system also allows the user to upload their own custom scripts, since the underlying service executes only binaries, a custom user script can be written in any automation language or software.

Network Configuration Context

The creation of different network configurations like Public IP and Private IP can be easy if we have a pool of public IPs to allocate. However, with our cost constraints, we limit ourselves to a single assigned to the gateway. Hence, in order to fool a bot into thinking that it has infected a machine with public IP, we implement a solution that is both low cost and efficient.

Our implementation of Public IP context involves a *two-stage NAT* (Public$\rightarrow$ Private$\rightarrow$ Public) strategy. In this setting, the execution machine (which has connectivity *only* through NAT on the Gateway) is assigned Public IP of our Gateway machine. We configure the *public vNIC* (Figure 4.1) with a NAT configuration, where it assigns the public IP of the Gateway (communicated by the system manager) to the infected machine. The NAT on the Public vNIC translates the Gateway IP (208.X.X.X in Figure 4.1) to the private IP

[1]This image was initially made using TRK as well

assigned to the physical NIC of the VM execution platform (10.1.4.10 in Figure 4.1) by the Gateway NAT. When the public IP context is required, the other private vNIC is removed by the system manager (through the hypervisor), and the opposite is done when a normal private IP context is required.

Iteration Management

The system manager is also required to chaperon, for every operational context family, the three minimum iterations in Titan.

After initializing the logging and containment modules, the system manager provides a whitelist of flows registered during an uninfected execution of our base-image with the user-emulation scripts. The system manager is then idle for the observation period, after which it stops the logging and containment module. Before reseting the VM, it uses nmap [38] to scan for any new opened ports (obviously, by the bot) and logs this information to help create fingerprints. It also updates the whitelist with all flows whose handshakes were allowed by the containment module in the previous iteration (details in next Section 4.1.2). This updated list is supplied in the new iteration where the manager then recreates the same context for a new cloned VM or for the baremetal machine.

4.1.2 Containment and Logging Module

The implementation of the containment logic is enforced by the POX controller [39] using network as well as application layer Deep Packet Inspection (DPI). We configure Open vSwitch [40] for full-packet (not just header) forwarding to the POX controller. The learning modules at the controller forces the virtual switch to either forward or drop the packet on the basis of its current rule set and the output of the attack sensors. The current rule set is maintained by the traffic classification module. There is also a DNS spy module that keeps records of queries in order to identify any domain generation algorithm of a bot.

We now describe our active sensors, traffic classification, dns spy and learning modules. We also describe our methodology for containment in a typical experiment of Titan.

Active Sensors

All outgoing traffic from the execution network first goes through active sensors, deployed inside POX, that actively block and log malicious traffic generated by the bot. A key insight to the effectiveness of our attack sensors is the following: we *know* that all traffic beyond our whitelist is malicious; our purpose then is just to prevent only that malicious traffic that can constitute an attack or the proliferation of the bot binary itself. Thus we have a sensor to detect the forwarding of an executable (`exec_detect`), and five attack and proliferation detection sensors.

We detect any DoS attacks launching from the infected virtual machine by tracking outgoing TCP SYN packets to every website. We block these packets if their count exceeds a threshold of 10 packets for any websites not visited by our user-emulation scripts. A bot may also target social or banking websites that our user emulation scripts visit; for these websites, we choose a threshold just above the attempts made (by the emulation scripts) in a previous clean run of the system. We also detect all TCP packets with a spoofed IP, as IP spoofing is often used to launch a DoS attack (`DoS_detect`).

We detect TCP SYN-scanning attempts by maintaining a list of IPs against a particular port (horizontal) and ports against a particular IP (vertical). Similar to Snort [41], we declare a scan if the number of SYN packets against a specific IP or port crosses a threshold value of 25 within a 2 minute time period (`netscan_detect`).

Any outgoing email from the infected machines is classified as spam. We inspect each outgoing packet for common HTTP web mail as well as SMTP message headers to trigger this classification. We also block communication on common SMTP ports (`spam_detect`).

The payload of any outgoing TCP packet is also scanned for SQL queries; any combination of `SELECT`,`FROM` & `WHERE` is flagged as an SQL injection attack (`inject_detect`).

We detect information stealing by inspecting each outgoing packet for transfer of user credentials (used by our user emulation scripts) in clear text (`info_detect`).

Traffic Classification

This module maintains records of IPs, URLs and ports. There are different types of lists that help other learning module in deciding the class of traffic. This module has the following lists: whitelist, blacklist, new_trafic, internal_traffic, dns_traffic, previous_traffic and cc_traffic. Table 4.1 shows the details of each list.

DNS Spy

This module is build upon the existing DNS spy module provided by POX controller. The built-in module spies on all DNS queries and keeps a record

Table 4.1: Traffic classification lists

List name	Purpose
Whitelist	Traffic generated by User activity emulation scripts
Blacklist	Traffic classified as attack by containment sensors
DNS_traffic	DNS queries towards our ISP DNS server
Internal_traffic	Traffic internal to execution network
New_traffic	New traffic generated by a bot but not seen previously
Previous_traffic	Traffic generated by a bot in previous iterations
cc_traffic	Requests destined towards known C&C servers

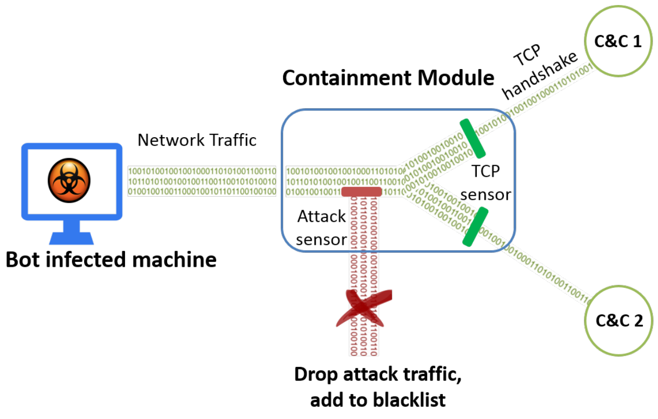
of questions and answers of each successful query. These records helps us in identifying sites with multiple IP addresses against a single domain name/URL. We modify this module by keeping record of failed queries as well. This helps us in identifying presence of domain generation algorithm (DGA) in a bot binary. When a bot infected machine has more than 5 failed DNS queries in a single iteration of an experiment, we identify this behavior as domain generation.

L2 Learning

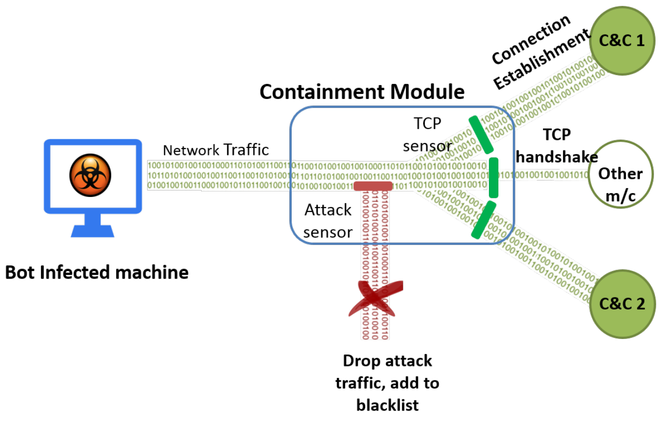
The L2 learning module decides whether to allow or drop a packet generated from execution machine. This module is based on traditional L2 learning component provided by POX controller. It queries traffic classification and Attack sensors to decide the fate of a packet. It is different from traditional learning module in the sense that it does not install flow rules against each allow or block decision. Since no flow rule is installed, the learning module constantly makes decision for every packet. It logs that decision along with its attack and IP/URL classification status of destination.

Containment Methodology

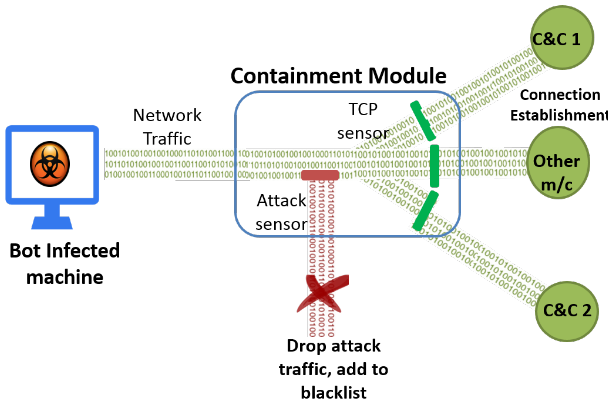
The containment rule-set is a combination of a white-list (WL) and a black-list (BL) of flows updated on each iteration of an experiment. The initial



(a) Containment: First Iteration



(b) Containment: Second Iteration



(c) Containment: Third Iteration

Figure 4.2: Different Iterations of Containment Module

WL consists of flows and domain names[2] extracted from a benign run, while the BL, initially empty, is dynamically populated by our active sensors. We use domain names in the WL because certain benign traffic, like visits to popular sites with CDNs, results in a different IP resolution every time. We allow DNS traffic for name resolution as we consider such traffic to pose no legal or ethical dilemma.

In every iteration we allow all traffic in our WL, block all traffic in the BL, and allow only the initial TCP hand-shake for any new traffic not flagged by our active sensors. There are two benefits in allowing such a TCP handshake. First, the bot binary is made to believe that it can contact any TCP-based C&C server around the globe with unfettered access to the Internet. Second, the bot allows us to monitor any built-in backup contact mechanism, while also revealing any IP and domain fluxing nature. Our policy generation logic then automatically augments the current WL with all such flows (new in current iteration and not blacklisted) to be allowed in the next iteration.

Thus, the **first iteration** allows us to monitor connections to non-WL flows, which are the bootstrapping connections attempted by the bot(Figure 4.2(a)). The **second iteration** will allow such bootstrapping flows to be established, but blocks any resulting new flow (beyond an initial handshake), that has a slight chance of causing attack or proliferation(Figure 4.2(b)). The **final iteration** will then allow this second round of post-bootstrapping flows, but again blocks any subsequent flows, most likely containing attack payloads. We stop the automation of our scripts at this point as our sensor heuristics become unreliable henceforth(Figure 4.2(c)). Instead, we provide those flows which our automated policy would have whitelisted in the next iteration, and were not black-listed by our active sensors, to an advanced user to manually accept a subset which augments the WL for another iteration with a more liberal policy.

Extensibility

The containment module implemented in Titan is extensible and easy to implement. The creation of new module involves writing a program in Python that interacts with the controller or other modules. Any custom module written by user can be added into the system during one of its iterations. Typically, a user can provide an attack sensor (e.g. examining traffic for XSS attacks) and specify the output file to be displayed along with the fingerprint. This feature of Titan allows the users to record new malware behaviors with minimal effort.

[2]Thus our flow definition consists of the {dest IP, dest port, URL} tuple

4.1.3 Fingerprint Generation and Presentation

Figure 4.3 shows the features of a bot fingerprint that Titan generates. The fingerprint generation engine works in two phases after an experiment completes.

In the first phase we process raw traffic for each operational context to generate context-specific features (those with parenthesis in Figure 4.3). The attack features are captured directly from the final blacklist and logs. Most of the C&C-related features (traffic volume, connection frequency, evasion, and C&C type) are generated by running simple heuristics over the captured pcap using Bro [42]. Other features are extracted by using additional tools. For example, URL features of C&C domains are generated by using a google safe browsing lookup API [43] and the requests library [44] in python . We discover the actual server running on a C&C port, if not well known, by using an nmap scan of that server. Finally, the geographical location of the C&C is extracted from the MaxMind [45] geo IP database using the pygeoip api [46].

Second, we generate meta-data for the experiment from its initial configuration. We then also look at differences between fingerprints from every different execution within the same context family to extract the context-specific behavior. This additional feature completes a faithful fingerprint. Currently we identify any change in network activity due to a change in execution platform as the *anti-analysis* feature. Furthermore, if the infected machine opens up additional ports when going from a private to public execution environment, we identify it as a potential *stepping-stone*. Finally, to populate the *information stealing* feature, we identify if the (fake) user-name and passwords used by our user-emulation scripts for logging into banking or social media sites are sent out by the bot. We are currently limited to only detecting such information sent by the bot in clear text. However, it is possible to employ heuristics to identify encrypted transfer of this information, by comparing with traffic from other operational contexts.

All these features are stored in an XML format and displayed on the same web frontend used to upload a binary and select the configuration parameters of the system. We build this website using an Apache server and a php backend.

4.2 Demonstration and Evaluation

We now present demonstration and evaluation of different features of Titan We begin by evaluating the containment methodology by studying Kanav F

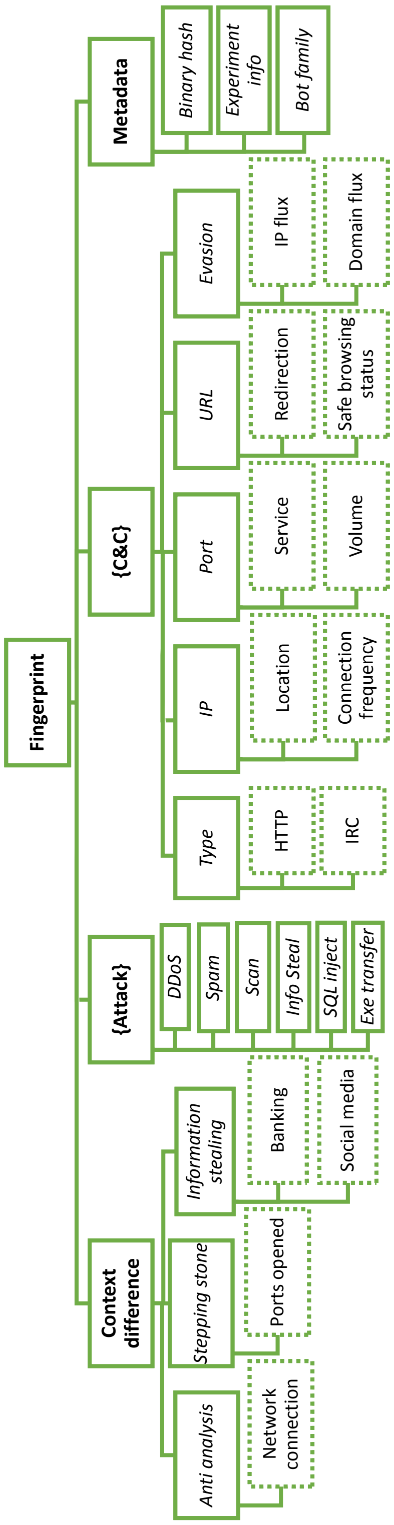


Figure 4.3: Features of a faithful bot fingerprint.

bot binary. Next, we deploy a custom Zeus botnet infrastructure in order to evaluate creation of different operational contexts. We also present evolution of cryptolocker botnet over the period of six months and detect changing behavior of bot. We then present fingerprint of Virut botnet demonstrating our different fingerprinting features.

4.2.1 Containment of Kanav.F

We now present a practical example demonstrating the containment methodology of Titan. We analyze binary of Kanav. F bot, this malware is known for downloading and installing several additional modules in the infected system.

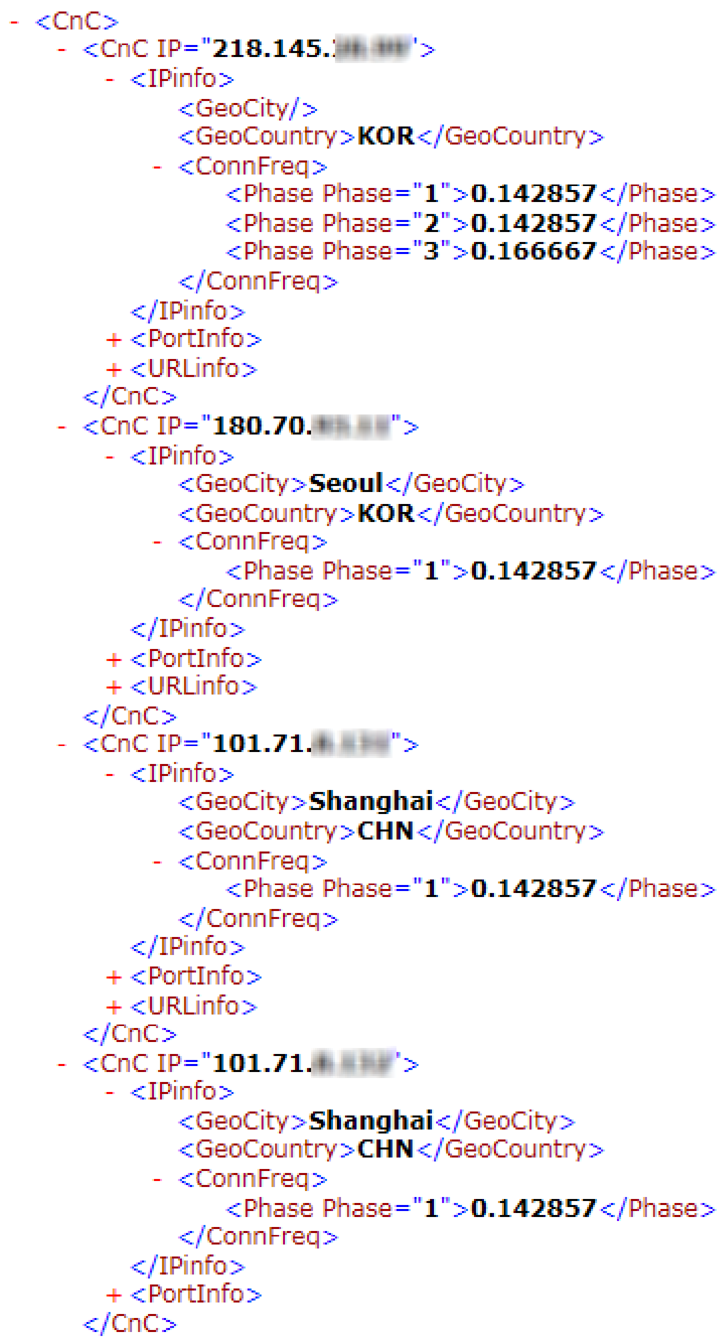
Experiment Methodology

We analyze this binary in three iterations of varying containment policy. The analysis of this binary is performed in default operational context of Titan. This context executes the bot binary in a virtual machine with NAT network configuration and no user activity. The time of all three iterations is varied from 5 to 30 minutes in order to capture complete network fingerprints of bot.

Results and Insights

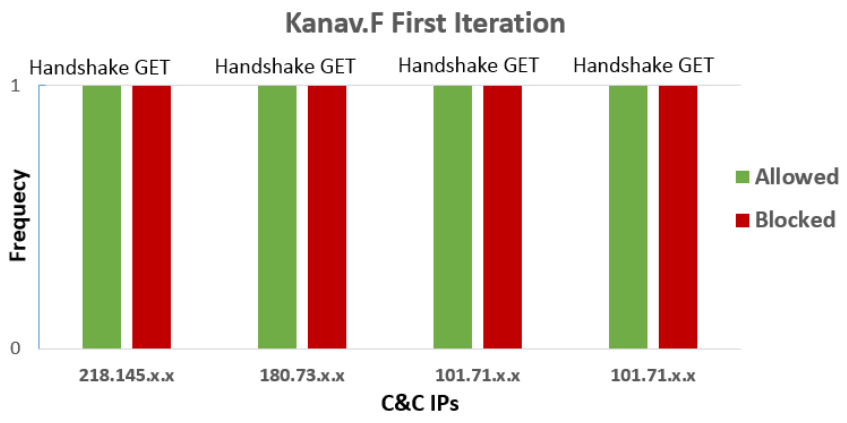
Figure 4.4 presents the C&C portion of Kanav F fingerprint generated by Titan. The important thing to observe in this fingerprint is the connection frequency against each C&C server. We see that the bot contacts C&C IP 218.145.x.x in all three iterations, whereas it only contacts rest of the three IPs in first iteration only. This corresponds to our theory that a bot, when denied complete access to the C&C server, will contact other C&C servers in the first iteration. In the second iteration, when it is granted access to its C&C server, it will not contact other/backup servers. Hence, due to our containment policy, we were able to observe bootstrapping behavior of Kanav.F.

In Figure 4.5(a) and 4.5(b), the frequency of Kanav F connection attempts is shown with respect to its C&C servers and containment module decisions. In the first iteration, presented in figure 4.5(a), we see that the bot contacts all of its C&C servers only once. The containment module allows the handshake and blocks the subsequent GET request made by the bot binary. However in the second iteration (figure 4.5(b)), we see that it only contacts just one of its C&C servers. Since we allow both handshake

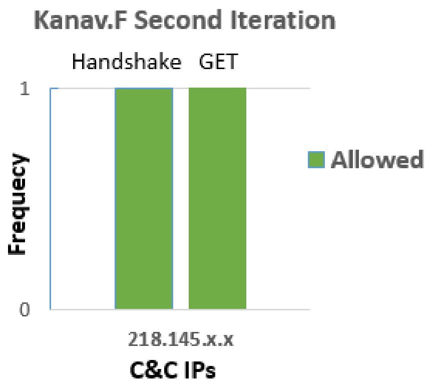


```
- <CnC>
  - <CnC IP="218.145.[redacted]">
    - <IPinfo>
        <GeoCity/>
        <GeoCountry>KOR</GeoCountry>
      - <ConnFreq>
          <Phase Phase="1">0.142857</Phase>
          <Phase Phase="2">0.142857</Phase>
          <Phase Phase="3">0.166667</Phase>
        </ConnFreq>
      </IPinfo>
    + <PortInfo>
    + <URLinfo>
    </CnC>
  - <CnC IP="180.70.[redacted]">
    - <IPinfo>
        <GeoCity>Seoul</GeoCity>
        <GeoCountry>KOR</GeoCountry>
      - <ConnFreq>
          <Phase Phase="1">0.142857</Phase>
        </ConnFreq>
      </IPinfo>
    + <PortInfo>
    + <URLinfo>
    </CnC>
  - <CnC IP="101.71.[redacted]">
    - <IPinfo>
        <GeoCity>Shanghai</GeoCity>
        <GeoCountry>CHN</GeoCountry>
      - <ConnFreq>
          <Phase Phase="1">0.142857</Phase>
        </ConnFreq>
      </IPinfo>
    + <PortInfo>
    + <URLinfo>
    </CnC>
  - <CnC IP="101.71.[redacted]">
    - <IPinfo>
        <GeoCity>Shanghai</GeoCity>
        <GeoCountry>CHN</GeoCountry>
      - <ConnFreq>
          <Phase Phase="1">0.142857</Phase>
        </ConnFreq>
      </IPinfo>
    + <PortInfo>
    </CnC>
```

Figure 4.4: C&C servers of Kanav F



(a) First Iteration Decision



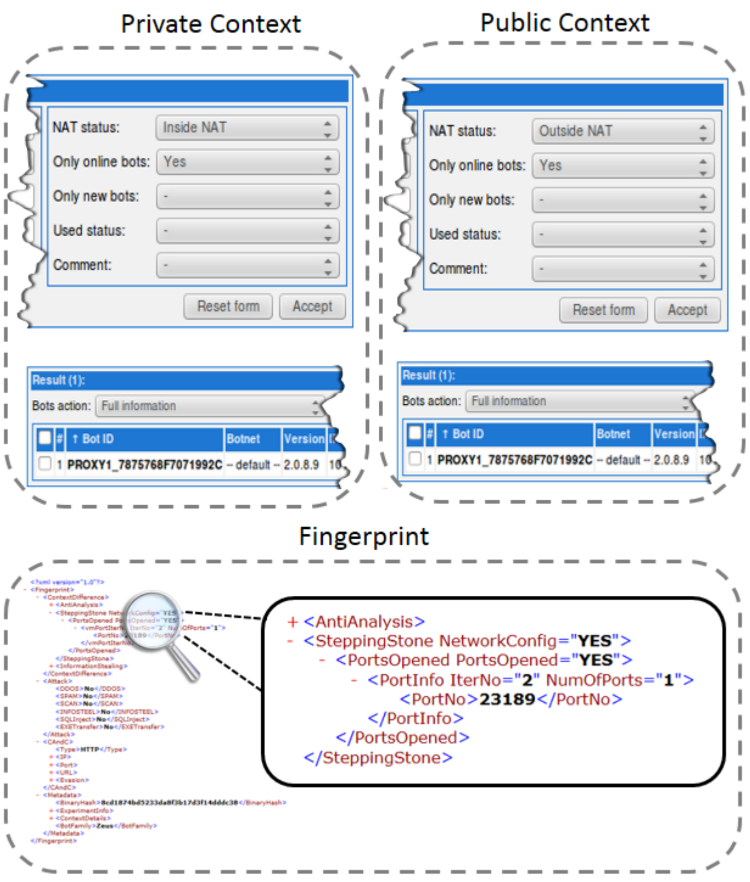
(b) Second Iteration Decision

Figure 4.5: Connection Frequency and Containment Decision of Kanav F

and GET request to go through, it does not contact other C&C servers. An important thing to observer here is that this C&C server (218.145.x.x) does not respond appropriately to the GET request. It returns a 404 not found, which essentially means that it has been taken down. This bot should now contact other C&C servers but due to the way it is programmed, it does not. While, we can only speculate about this behavior of bot, we see that our containment module immensely helps us in extracting this information and we can easily spot it in our fingerprint.

4.2.2 Case Study: Zeus

In order to verify the effectiveness of operational contexts, we deploy our own botnet infrastructure using the code for Zeus botnet [47]. Zeus is a famous



Private Context:

NAT status:	Inside NAT
Only online bots:	Yes
Only new bots:	-
Used status:	-
Comment:	-

Result (1):
Bots action: Full information

#	↑ Bot ID	Botnet	Version
1	PROXY1_7875768F7071992C	-- default --	2.0.8.9

Public Context:

NAT status:	Outside NAT
Only online bots:	Yes
Only new bots:	-
Used status:	-
Comment:	-

Result (1):
Bots action: Full information

#	↑ Bot ID	Botnet	Version
1	PROXY1_7875768F7071992C	-- default --	2.0.8.9

Fingerprint:

```
<?xml version="1.0"?>
- <Fingerprint>
  - <ContextDifference>
    + <AntiAnalysis>
    - <SteppingStone NetworkConfig="YES">
      - <PortsOpened PortsOpened="YES">
        - <vmPortIterNo IterNo="2" NumOfPorts="1">
            <PortNo>23189</PortNo>
          </vmPortIterNo>
        </PortsOpened>
      </SteppingStone>
    + <InformationStealing>
    </ContextDifference>
  - <Attack>
      <DDOS>No</DDOS>
      <SPAM>No</SPAM>
      <SCAN>No</SCAN>
      <INFOSTEEL>No</INFOSTEEL>
      <SQLInject>No</SQLInject>
      <EXETransfer>No</EXETransfer>
    </Attack>
  - <CAndC>
      <Type>HTTP</Type>
    + <IP>
    + <Port>
    + <URL>
    + <Evasion>
    </CAndC>
  - <Metadata>
      <BinaryHash>8cd1874bd5233da8f3b17d3f14dddc38</BinaryHash>
    + <ExperimentInfo>
    + <ContextDetails>
      <BotFamily>Zeus</BotFamily>
    </Metadata>
</Fingerprint>
```

```
+ <AntiAnalysis>
- <SteppingStone NetworkConfig="YES">
  - <PortsOpened PortsOpened="YES">
    - <PortInfo IterNo="2" NumOfPorts="1">
        <PortNo>23189</PortNo>
      </PortInfo>
    </PortsOpened>
  </SteppingStone>
```

Figure 4.6: Emulating different network contexts and behavior for the Zeus botnet

open source *do it yourself* botnet, known for stealing financial information of users. This bot has many variants present in the wild and its code is used as a basic building block for various other malwares. We first present implementation details of our botnet infrastructure and then our results of executing a custom binary inside Titan.

Zeus Infrastructure

Zeus source code is divided into two parts, malware maker and command and control setup. The malware maker code is used for compiling bot binary with custom parameters like list of C&C server IPs and URLs and list of websites to spy.

We position the C&C server on another machine (on a different subnet from our testbed) through which a botmaster can push arbitrary functionality in the form of executables that are then executed on the infected machine. Furthermore, we do not modify the C&C server code and only recompile the bot binary to provide the IP of our C&C server. The bot contacts our C&C server every 2 minutes to download a new configuration file and to send user activity reports. We also observe that the Zeus C&C server control panel displays network operational context (inside or outside NAT) for a victim machine, indicating the importance of this behavior for a botmaster.

Results and Insights

To evaluate the effectiveness of our tool, we run our system by varying the network context between Private/NAT and Public. Figure 4.6 shows that for both different execution scenarios, Titan fools the C&C logic into thinking that the same Bot ID is first inside and then outside of NAT. We then, for the outside NAT scenario, push onto the bot a simple port listening script, indicating that the node should act as a stepping-stone. This emulates a botnet that would want to use different network conditions for different purposes. We observe that the fingerprint Titan generates captures this change in behavior under the *stepping-stone* feature, as shown in the bottom half of Figure 4.6.

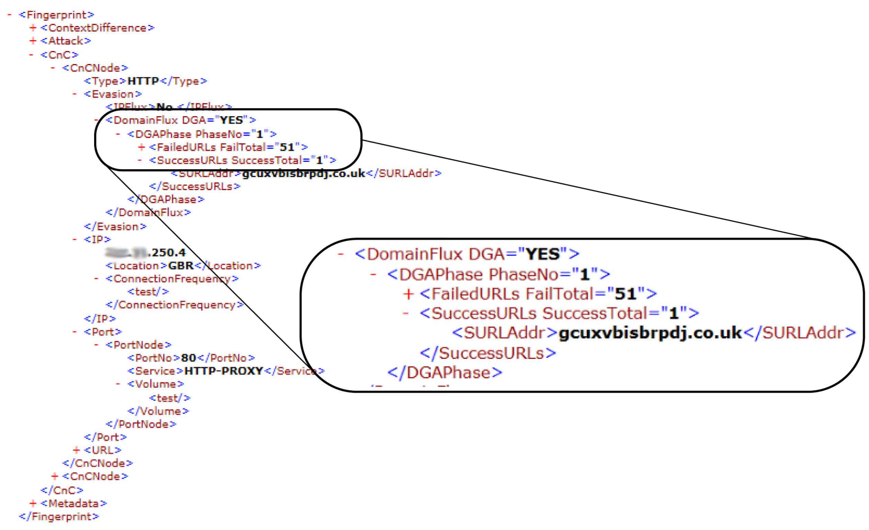
4.2.3 Evolution of Cryptolocker

To demonstrate fingerprint generation for an active botnet, we analyzed two different binaries of the `CryptoLocker` botnet, which besides its notoriety as a hard-to-remove ransomware, runs a known DGA algorithm [48]. We show that our system can easily identify the DGA behavior of this botnet. We also observe the evolution of this bot by analyzing a more recent binary. We show that DGA algorithm of this bot has been updated and it now carries the feature of evasion.

Analysis of Cryptolocker - December 2013 Version

We first present the results of Cryptolocker binary found in December 2013. Figure 4.7 highlights part of a resulting fingerprint, with a 5 minute observation time in the private network configuration. We observe 51 failed DNS queries and 1 successful resolution within the first 5 minutes of analysis. Our system correctly flags it as a bot with DGA ability, as shown in Figure 4.7

Further iterations of this binary reveal that one successfully resolved domain is being redirected to a sinkhole; hence the only interesting feature



```
- <Fingerprint>
   + <ContextDifference>
   + <Attack>
   - <CnC>
      - <CnCNode>
           <Type>HTTP</Type>
         - <Evasion>
              <IPFlux>No </IPFlux>
            - <DomainFlux DGA="YES">
               - <DGAPhase PhaseNo="1">
                  + <FailedURLs FailTotal="51">
                  - <SuccessURLs SuccessTotal="1">
                       <SURLAddr>gcuxvbisbrpdj.co.uk</SURLAddr>
                    </SuccessURLs>
                 </DGAPhase>
              </DomainFlux>
           </Evasion>
         - <IP>
              [redacted].250.4
              <Location>GBR</Location>
            - <ConnectionFrequency>
                 <test/>
              </ConnectionFrequency>
           </IP>
         - <Port>
            - <PortNode>
                 <PortNo>80</PortNo>
                 <Service>HTTP-PROXY</Service>
               - <Volume>
                    <test/>
                 </Volume>
              </PortNode>
           </Port>
         + <URL>
        </CnCNode>
      + <CnCNode>
     </CnC>
   + <Metadata>
  </Fingerprint>
```

```
- <DomainFlux DGA="YES">
   - <DGAPhase PhaseNo="1">
      + <FailedURLs FailTotal="51">
      - <SuccessURLs SuccessTotal="1">
           <SURLAddr>gcuxvbisbrpdj.co.uk</SURLAddr>
        </SuccessURLs>
     </DGAPhase>
```

Figure 4.7: `CryptoLocker 2013` fingerprint generated by Titan

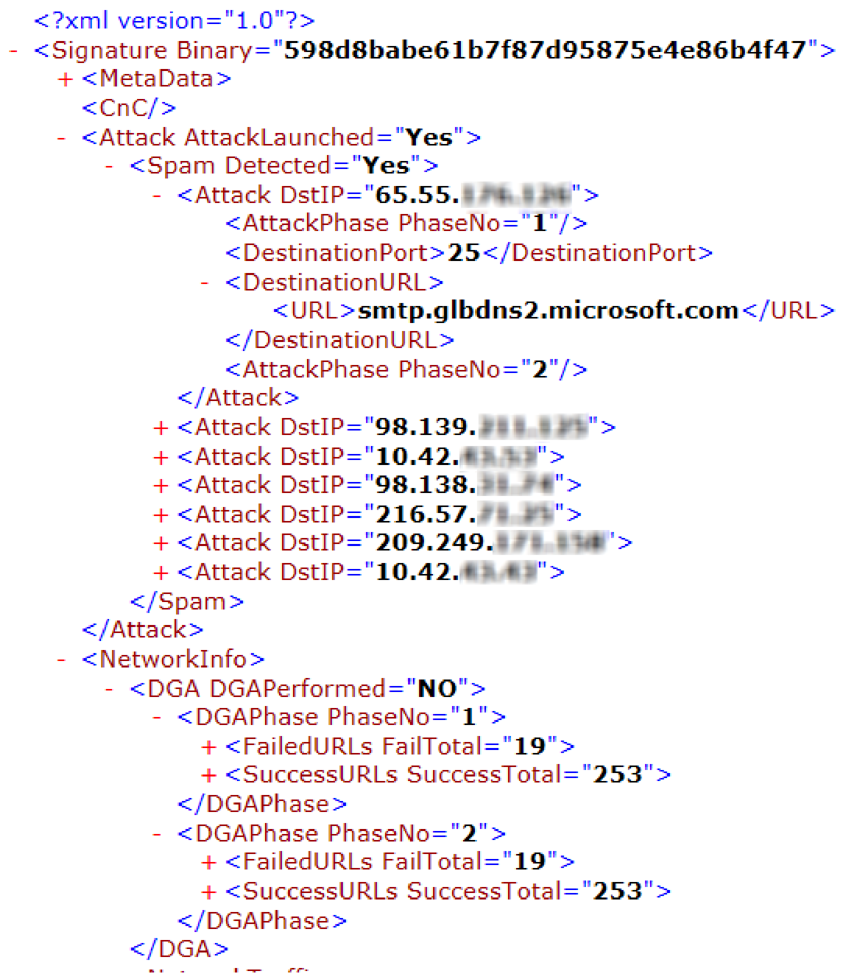
of this bot binary, i.e. DGA, is successfully detected and reported by our system.

Analysis of Cryptolocker - April 2014 Version

We now present the results of an updated Cryptolocker binary found in April 2014. Figure 4.8 highlights part of a resulting fingerprint. This binary is tested in the same conditions as the December 2013 version. We observe 253 successful DNS queries and 19 failed ones in 5 minute analysis. Some of these queries are also towards known SMTP servers and our system flags it as the bot trying to send spam emails.

In this version of binary, we do not detect DGA feature due to the fact that now this bot tries to evade intrusion detection systems by initiating connections towards large number of benign websites.

Hence by observing two different variants of a bot, we were able to detect its change in behavior over time.



```
  <?xml version="1.0"?>
- <Signature Binary="598d8babe61b7f87d95875e4e86b4f47">
  + <MetaData>
    <CnC/>
  - <Attack AttackLaunched="Yes">
    - <Spam Detected="Yes">
      - <Attack DstIP="65.55.[redacted]">
          <AttackPhase PhaseNo="1"/>
          <DestinationPort>25</DestinationPort>
        - <DestinationURL>
            <URL>smtp.glbdns2.microsoft.com</URL>
          </DestinationURL>
          <AttackPhase PhaseNo="2"/>
        </Attack>
      + <Attack DstIP="98.139.[redacted]">
      + <Attack DstIP="10.42.[redacted]">
      + <Attack DstIP="98.138.[redacted]">
      + <Attack DstIP="216.57.[redacted]">
      + <Attack DstIP="209.249.[redacted]">
      + <Attack DstIP="10.42.[redacted]">
      </Spam>
    </Attack>
  - <NetworkInfo>
    - <DGA DGAPerformed="NO">
      - <DGAPhase PhaseNo="1">
        + <FailedURLs FailTotal="19">
        + <SuccessURLs SuccessTotal="253">
        </DGAPhase>
      - <DGAPhase PhaseNo="2">
        + <FailedURLs FailTotal="19">
        + <SuccessURLs SuccessTotal="253">
        </DGAPhase>
      </DGA>
```

Figure 4.8: `CryptoLocker 2014` fingerprint generated by Titan

Chapter 5

Conclusion and Future Work

This chapter presents summary of contributions of this thesis (section 5.1), key findings and future work in this area (section 5.2 & 5.3).

5.1 Summary of Contributions

In this thesis, our aim was to design and implement various features of a botnet analysis tool which is opensource, low cost, easy to setup and manage. It examines bot binaries using semi automation of containment policies and under various operational contexts. It is extensible and generates network fingerprints in a hierarchical and user friendly format.

The first contribution of this thesis is methodology for semi-automation of containment policies. We present an iterative approach to containment in chapter 3. We then present the implementation of this approach using Software Defined Networking and demonstrate the efficacy of containment using real world bot in chapter 4.1.

The second contribution of this thesis is identification and implementation of various operational context in which a bot shows its behavior. We identify these context as well as various factors that affect them in chapter 3. We present the implementation details of three basic contexts in chapter 4.1 and deploy our own botnet infrastructure in order to evaluate its effectiveness of this approach.

The third contribution of this thesis is presentation of an extensible and easy to setup and manage botnet analysis tool. We present in chapter 4.1 the implementation details of this tool and discuss various components that make it easy to manage as well as extensible.

5.2 Conclusions

We now conclude by presenting key finding of this thesis.

Semi-automated iterative refinement of containment policies
We first examined existing approaches of containment in malware analysis and then proposed an iterative mechanism for performing containment of harmful activities in botnet analysis. The proposed method is also semi-automated, thus requiring minimal human intervention. Our approach is to allow understood activity iteratively. We start of with a conservative policy of allowing no outside connections beyond TCP handshake and also block any traffic marked as an attack by our sensors. This approach helps us in capturing the bootstrap behavior of a bot. We then allow all non-attack standard TCP traffic to pass through to the Internet. This is done using Software Defined Networking, we use POX controller along with Open vSwitch in our network. The traffic generated by a bot is constantly monitored by attack sensors and all the forwarding decisions are made by our learning module. The containment policies are revised in every iteration and they gradually move towards a more liberal rule set. Our sensors are extensible and any user can add their POX based monitoring and blocking modules in our system.

Capturing changing bot behavior
We first identify various operational contexts that affect a behavior of a bot. These contexts include user activity in infected machine, type of infected machine, and network configuration of infected machine. We then identify various factors like observation duration and containment policies that affect these contexts. We present novel ways in which these contexts can be used by botnet analysis tools. We also present their implementation in our own botnet analysis tool called Titan. We show these contexts are effective by deploying a real world botnet infrastructure and fooling its operator.

5.3 Future Work

While a lot of work has been done in developing this tool, there are some features that can be enhanced. Firstly, the attack sensors are limited and the tool needs more breadth of attacks to be covered. Secondly, the operational contexts that are implemented are also somewhat limited, if they are increased, they would surely capture complete network behavior of bot.

Bibliography

[1] Brett Stone-Gross, Marco Cova, Lorenzo Cavallaro, Bob Gilbert, Martin Szydlowski, Richard Kemmerer, Christopher Kruegel, and Giovanni Vigna. Your botnet is my botnet: Analysis of a botnet takeover. In *Proceedings of the 16th ACM Conference on Computer and Communications Security*, CCS '09, pages 635–647, New York, NY, USA, 2009. ACM.

[2] Zhen Li, Qi Liao, and Aaron Striegel. Botnet economics: Uncertainty matters. In *WEIS'2008*, 2008.

[3] Yinglian Xie, Fang Yu, Kannan Achan, Rina Panigrahy, Geoff Hulten, and Ivan Osipkov. Spamming botnets: Signatures and characteristics. In *Proceedings of the ACM SIGCOMM 2008 Conference on Data Communication*, SIGCOMM '08, pages 171–182, New York, NY, USA, 2008. ACM.

[4] Trendmicro. `http://goo.gl/eMNDio`.

[5] Christian Kreibich, Nicholas Weaver, Chris Kanich, Weidong Cui, and Vern Paxson. Gq: Practical containment for measuring modern malware systems. In *Proceedings of the ACM IMC*, pages 397–412, New York, NY, USA, 2011. ACM.

[6] John P. John, Alexander Moshchuk, Steven D. Gribble, and Arvind Krishnamurthy. Studying spamming botnets using botlab. In *Proceedings of the NSDI*, 2009.

[7] Moheeb Abu Rajab, Jay Zarfoss, Fabian Monrose, and Andreas Terzis. A multifaceted approach to understanding the botnet phenomenon. In *Proceedings of the ACM IMC*, 2006.

[8] Malwr. `https://malwr.com/`.

[9] Anubis. `http://anubis.iseclab.org/`.

[10] Mcafee labs threats report: February 2015. `http://www.mcafee.com/us/resources/reports/rp-quarterly-threat-q4-2014.pdf`.

[11] Symantec internet security threat report. `http://www.symantec.com/security_response/publications/threatreport.jsp`.

[12] Carsten Willems, Thorsten Holz, and Felix Freiling. Toward automated dynamic malware analysis using cwsandbox. *IEEE Security and Privacy*, pages 32–39, March 2007.

[13] Chris Kanich, Christian Kreibich, Kirill Levchenko, Brandon Enright, Geoffrey M. Voelker, Vern Paxson, and Stefan Savage. Spamalytics: An empirical analysis of spam marketing conversion. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pages 3–14, New York, NY, USA, 2008. ACM.

[14] Ulrich Bayer, Imam Habibi, Davide Balzarotti, Engin Kirda, and Christopher Kruegel. A view on current malware behaviors. In *Proceedings of the LEET'09*, LEET'09, pages 8–8, Berkeley, CA, USA, 2009. USENIX Association.

[15] Yuanyuan Zeng, Xin Hu, and K.G. Shin. Detection of botnets using combined host- and network-level information. In *Dependable Systems and Networks (DSN), 2010 IEEE/IFIP International Conference on*, pages 291–300, June 2010.

[16] Jestin Joy and Anita John. Host based attack detection using system calls. In *Proceedings of the Second International Conference on Computational Science, Engineering and Information Technology*, CCSEIT '12, pages 7–11, New York, NY, USA, 2012. ACM.

[17] Anestis Karasaridis, Brian Rexroad, and David Hoeflin. Wide-scale botnet detection and characterization. In *Proceedings of the First Conference on First Workshop on Hot Topics in Understanding Botnets*, HotBots'07, pages 7–7, Berkeley, CA, USA, 2007. USENIX Association.

[18] Heng Yin, Dawn Song, Manuel Egele, Christopher Kruegel, and Engin Kirda. Panorama: Capturing system-wide information flow for malware detection and analysis. In *Proceedings of the 14th ACM Conference on Computer and Communications Security*, CCS '07, pages 116–127, New York, NY, USA, 2007. ACM.

[19] Artem Dinaburg, Paul Royal, Monirul Sharif, and Wenke Lee. Ether: Malware analysis via hardware virtualization extensions. In *Proceedings of the 15th ACM Conference on Computer and Communications Security*, CCS '08, pages 51–62, New York, NY, USA, 2008. ACM.

[20] Dawn Song, David Brumley, Heng Yin, Juan Caballero, Ivan Jager, Min Gyung Kang, Zhenkai Liang, James Newsome, Pongsin Poosankam, and Prateek Saxena. Bitblaze: A new approach to computer security via binary analysis. In *Proceedings of the 4th International Conference on Information Systems Security*, ICISS '08, pages 1–25, Berlin, Heidelberg, 2008. Springer-Verlag.

[21] Clemens Kolbitsch, Paolo Milani Comparetti, Christopher Kruegel, Engin Kirda, Xiaoyong Zhou, and XiaoFeng Wang. Effective and efficient malware detection at the end host. In *Proceedings of the 18th Conference on USENIX Security Symposium*, SSYM'09, pages 351–366, Berkeley, CA, USA, 2009. USENIX Association.

[22] Cwsandbox. `http://mwanalysis.org/`.

[23] Christian Rossow, Christian J. Dietrich, Herbert Bos, Lorenzo Cavallaro, Maarten van Steen, Felix C. Freiling, and Norbert Pohlmann. Sandnet: Network Traffic Analysis of Malicious Software. In *Proceedings of ACM BADGERS Workshop*, 2011.

[24] Guofei Gu, Phillip Porras, Vinod Yegneswaran, Martin Fong, and Wenke Lee. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *Proceedings of 16th USENIX Security Symposium*, Berkeley, CA, USA, 2007. USENIX Association.

[25] Guofei Gu, Roberto Perdisci, Junjie Zhang, and Wenke Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proceedings of the 17th Conference on Security Symposium*, SS'08, Berkeley, CA, USA. USENIX Association.

[26] Fireeye. `http://www.fireeye.com/products-and-solutions/`.

[27] Damballa. `https://www.damballa.com/`.

[28] None. `http://camas.comodo.com/`. Online. December,2013.

[29] Pavan Roy Marupally and Vamsi Paruchuri. Comparative analysis and evaluation of botnet command and control models. In *AINA*, pages 82–89. IEEE Computer Society, May 2010.

[30] Thorsten Holz, Moritz Steiner, Frederic Dahl, Ernst Biersack, and Felix Freiling. Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm. In *Proceedings of the LEET 2008*, LEET'08, pages 9:1–9:9, Berkeley, CA, USA. USENIX Association.

[31] H. Binsalleeh, T. Ormerod, A. Boukhtouta, P. Sinha, A. Youssef, M. Debbabi, and L. Wang. On the analysis of the zeus botnet crimeware toolkit. In *Eighth Annual International Conference on Privacy Security and Trust (PST)*, 2010.

[32] G Kirubavathi and R Anitha. Botnets: A study and analysis. In *Computational Intelligence, Cyber Security and Computational Models*, pages 203–214. Springer, 2014.

[33] S. Khattak, N. Ramay, K. Khan, A. Syed, and S. Khayam. A taxonomy of botnet behavior, detection, and defense. *IEEE Commun. Surveys Tuts.*, PP(99):1–27, Oct 2013.

[34] Dionaea. `http://dionaea.carnivore.it/`.

[35] Paul Barham, Boris Dragovic, Keir Fraser, Steven Hand, Tim Harris, Alex Ho, Rolf Neugebauer, Ian Pratt, and Andrew Warfield. Xen and the art of virtualization. In *Proceedings of the ACM SOSP*, SOSP '03, pages 164–177, New York, NY, USA, 2003. ACM.

[36] Trinity rescue kit. `http://trinityhome.org/`.

[37] Autoit. `http://www.autoitscript.com/site/autoit`.

[38] Nmap security scanner. `http://nmap.org/`.

[39] Pox. `http://www.noxrepo.org/pox/about-pox/`.

[40] Ben Pfaff, Justin Pettit, Teemu Koponen, Keith Amidon, Martin Casado, and Scott Shenker. e.a.: Extending networking into the virtualization layer. In *In: 8th ACM Workshop on Hot Topics in Networks*, October 2009.

[41] Martin Roesch. Snort - lightweight intrusion detection for networks. In *Proceedings of the 13th USENIX Conference on System Administration*, LISA, pages 229–238, Berkeley, CA, USA, 1999. USENIX Association.

[42] Bro. `http://www.bro.org,`.

[43] Google safe browsing lookup api. `http://goo.gl/JnV8Fk`.

[44] Requests. `http://requests.readthedocs.org`.

[45] Maxmind. `http://www.maxmind.com`.

[46] Pygeoip. `https://github.com/appliedsec/pygeoip`.

[47] Visgean skeloru. `https://github.com/Visgean/Zeus`.

[48] Securelist. `http://goo.gl/vC1JCQ`.