ISLAMIC UNIVERSITY OF TECHNOLOGY(IUT)
Department of Computer Science and Engineering

Azizul Hassan Arif
Student Id: 094412
Computer Science & Engineering
Islamic University of Technology
Board Bazar, Gazipur
E-mail: arifcit@iut-dhaka.edu

Nahid Ebna Hasan
Student Id: 094427
Computer Science & Engineering
Islamic University of Technology
Board Bazar, Gazipur
E-mail: nahidcit@iut-dhaka.edu

BotEliminator: Detecting Botmaster & Botnet Using IP Traceback Mechanism

B.Sc Thesis

Supervisor: Md. Shakhawat Hossen
Assistant Professor
Computer Science & Engineering
Islamic University of Technology
Board Bazar, Gazipur
E-mail: shakhawat@iut-dhaka.edu

Acknowledgement

At first we would like to thank our respected supervisor Shakhawat Hossen, Assistant Professor,CSE Department for his guidence, supervision and earnest support through out the whole thesis work.We are grateful to our honorable Dr. Muhammad Mahbub Alam, Associate Professor, CSE Department, for his guidance and co-operation. We thank all the teacher of CSE department for their support and enthusiasm.

Azizul Hassan Arif

Nahid Ebna Hasan

Table of Contents

List of Figures

Abstract

Internet security is an important issue in our modern life.Todays Internet security has been threatened by many malicious activities.BOTs are now recognized as one of the most serious security threat.Because they are responsible for Data theft,Dos attack,Click fraud and many more.Many Botnet detection methods have been proposed to mitigate the threat.
In this Paper, we proposed a mechanism for detecting botnets as well as botmaster using IP traceback mechanism.Mainly botmaster is the root of the whole botnet system.So detection only of botnet mitigate a portion of threat,while detection of botmaster mitigate the full.

Chapter 1

Introduction

1.1 Background

The growing reliance on the Internet has introduced numerous challenges to the protection of the privacy, integrity and security of user data. Although convenient, the use of Internet-based services poses a number of security challenges. The main security threat and the main carrier of malicious activities on the Internet is malicious software, also known as malware. Malware implements a variety of malicious and illegal activities that disrupt the use of a compromised computer and jeopardize the security of the user's data. Over the last decade, malicious software or malware has risen to become a primary steric of most of the scanning, data theft, click fraud, (distributed) denial-of-service (DOS) activities, and direct attacks, taking place across the Internet. Among the various forms of malicious software, botnets in particular have recently distinguished themselves to be among the premier threats to computing assets.

1.2 What is BOT,BOTNET & BOTMASTER

1.2.1 BOT:

Bot comes from the word robot, an automated system.Software application that can run automated tasks over the Internet is called Bot.It's a computer system or malicious software component that can be remotely controlled by an attacker. Bots are implemented where response speed is faster than that of humans is required.

1.2.2 BOTNET:

Coordinated group of compromised host connected with command & control channel is called Botnet.Botnet is controlled by a "botmaster" and utilized as "restheirce" or "platform" for attacks such as distributed denial-of-service (DDoS) attacks, and fraudulent activities such as spam, phishing, identity theft, and information exfiltration etc.

1.2.3 BOTMASTER:

The malicious software components are coordinated over command & control channel by a single entity is called the botmaster.

1.3 Command & Control Channel(C&C channel)

The main carrier of botnet functionality and the defining characteristic of bot malware is called Command & Controll Channel (C&C Channel).It represents a communication channel established between the botmaster and compromised hosts.This channel is used by the attacker to issue commands to bots and receive information from the compromised machines.The C&C channel enables remote coordination of a large number of bots, and it introduces the level of flexibility in botnet operations by creating the ability to change and update malicious botnet code.

1.4 Botnet Architecture

On the basis of topology of the C&C network, botnets can be classified as botnets with three types of network architecture.

1. Centralized.
2. Decentralized.
3. Hybrid.

Centralized botnets have centralized C&C network architecture, where all bots in a botnet contact one or several C&C servers owned by the same botmaster. Centralized C&C channels can be realized using various communication protocols, such as IRC (Internet Relay Chat), HTTP and HTTPS.

IRC-based botnets are created by deploying IRC servers or by using IRC servers in public IRC networks. In this case, the botmaster specifies a chat channel on a IRC server to which bots connect to in order to receive commands. This model of operation is referred to as the push model, as the botmaster "pushes" commands to bots. HTTP-based botnets are another common type of centralized botnets, that rely on HTTP or HTTPS transfer protocol to transfer C&C messages. In contrast to bots in IRC-based botnets, bots in HTTP-based botnets contact a theyb-based C&C server notifying their existence with system-identifying information via HTTP or HTTPS requests. As a response, the malicious server sends back commands or updates via counterpart response messages. This model of operation is referred to as the pull model, as bots have to "pull" the commands from the centralized C&C server. The IRC-and and the HTTP-based botnets are easy to deploy and manage, and they are very eficient in implementing the botmasters' malicious agenda, due to the low latency of command messages. For this reason, the IRC-based and the HTTP-based C&C have been widely used for deploying botnets. Hotheyver, the main drawback of botnets with centralized network architecture is that they are vulnerable to the single point of failure. That is, once the C&C servers have been identified and disabled, the entire botnet could be taken down. Decentralized botnets represent a class of botnets developed with the goal of being more resilient to neutralization techniques. Botnets with decentralized C&C infrastructure have adopted P2P (Peer-to-Peer) communication protocols as the means of communicating within a botnet. This implies that bots belonging to the P2P botnet form an overlay network in which the botmaster can use any of the bots (P2P nodes) to distribute commands to other peers or to collect information from them In these botnets, the botmaster can join and issue commands at any place or time P2P botnets are realized either by using some of the existing P2P transfer protocols, such as Kademlia, Bittorent and Overnet, or by custom P2P transfer protocol. While more complex and perhaps more costly to manage and operate compared to centralized botnets, P2P botnets offer high resiliency, since even if the significant portion of the P2P botnet is taken down the remaining bots may still be able to communicate with each other and with the botmaster, thus pursuing their malicious purpose. Hotheyver, P2P botnets have one major drawback. They cannot guarantee high reliability and low latency of C&C communication, which severely limits the overall eficiency of orchestrating attacks. Some of the recent botnets have adopted more advanced hybrid network architectures that combine principles of centralized and decentralized C&C network architectures. This class of botnets uses advanced hybrid P2P communication protocols that try to combine resiliency of P2P botnets with the low latency of communication

of centralized botnets. The hybrid botnet architecture has been investigated by several groups of authors, such as Wang et al. and Z. Zhang et al. The authors suggest that in order to provide both resilience and low latency of communication hybrid botnets should be realized as networks in which bots are interconnected in P2P fashion and organized in two distinct groups: the group of proxy bots and the group of working bots. Working bots would implement the malicious activity while proxy bots would provide the propagation of C&C messages from and to the botmaster. Working bots would periodically connect to the proxy bots in order to receive commands. Based on the work presented in this topology should provide a resilience to take down efforts as well as improvements in latency of C&C messages comparing to the regular P2P botnets.

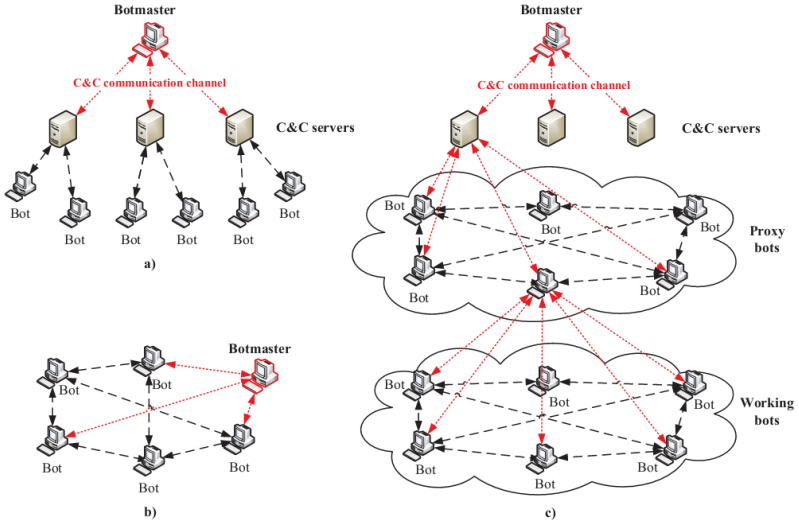

Figure 1.1: Botnet Architecture

1.5 Botnet Life Cycle

Botnet operation can be addressed through the analysis of botnet life-cycle i.e. the set of bot's functional phases observable during the botnet operation. The detection approaches target speci?c phases of botnet life-cycle, by utilizing speci?c heuristics of botnet behavitheir within these phases. Therefore, the understanding of the botnet life-cycle is crucial to the successful analysis

of the existing work on botnet detection. The botnet operation divided into three distinct phases:

1. The infection phase.
2. The communication phase
3. The Attack phase.

The first phase of the botnet life-cycle is the Infection phase in which vulnerable computers are compromised by the bot malware, thus becoming zombies within a specific botnet. Usually this phase can be further divided into two sub-phases known as Initial Infection and Secondary Infection. During the initial infection sub-phase computers are infected by malicious piece of software known as a "loader". The initial infection can be realized in different ways, for instance, through the unwanted download of malware from malicious websites, through the download of infected files attached to email messages, by propagation of malware from infected removable disks, etc. The loader primary role is to assist in obtaining the bot malware binary. Upon successful initial infection, the secondary infection sub-phase start, during which the loader downloads the malware binary from an external network location and installs in on the vulnerable machine. The bot malware binaries can be downloaded using diverse protocols, such as FTP (File Transfer Protocol), HTTP (Hypertext Transfer Protocol) / HTTPS (Hypertext Transfer Protocol Secure) or some of the P2P (Peer-to-Peer) transfer protocol.

The second phase of the botnet life-cycle the Communication phase. This phase includes several botnet operational modes that entail communication between compromised computers and C&C servers. The communication phase covers communication devoted to receiving instructions and updates from the botmaster, as well as the reporting on the current status of bots. The communication covers several modes of operation: initial connection attempts to the C&C server upon successful infection phase, connection attempts by the bot after reboot of the compromised machine, periodical connection attempts in order to report the status of the infected machine, as well as the connection attempts initiated by the C&C server in order to update malware code or propagate instructions to bots. The communication between zombie and the C&C server is realized using the C&C channel that can be implemented in different ways.

The third phase of botnet life-cycle is marked as Attack phase as it includes bot operation aimed at implementing attackers' malicious agenda. During

attack phase zombie computer may launch DDoS attacks, start SPAM e-mail campaigns, perform distribution of the stolen identities, deploy click-fraud, manipulate online reputation systems and surveys, etc. In this operational phase the bots can also implement propagation mechanism, such as scanning for vulnerable computers or distributing malicious software. The second and the third phase are functionally linked so they are usually altering one after another, once a vulnerable computer is successfully infected. However it should be noted that different phases within the botnet life-cycle can last for different time spans, and that the length of a specific phase can vary depending on the attack campaign the bot implements.

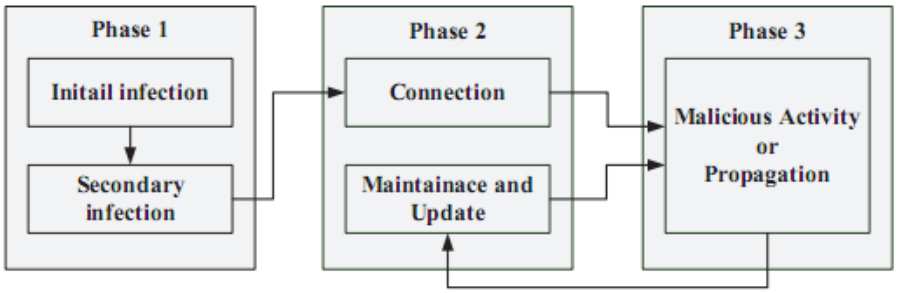

Figure 1.2: Botnet Life Cycle

1.6 Problems with Botnet

Botnets are utilized to perform the following:

1. Distributed Denial-of-Service Attacks.
2. Spam
3. Click Fraud
4. Data Theft etc.

1.7 Why Botnet Detection is Difficult?

Botnet C&C traffic is difficult to detect for the following:

1. C&C traffic follows normal protocol similar to normal traffic (i.e. IRC,HTTP).
2. Potentially encrypted C&C traffic
3. Fake benign connections
4. There may be very few bots in the monitored network
5. Traffic volume is low.

1.8 Botnet Detection Methods

Based on bot behavior and traffic patterns, detection approaches are two types:

1. Network Based Detection Method.
2. Client Based Detection Method.

1.8.1 Network based detection method

Network-based detection is based on analysis of network trafic in order to identify presence of compromised computers. This class of detection methods detects botnets by identifying trafic produced by botnets operating in all three phases of botnet life-cycle. The trafic is usually analyzed on either packet level or low level. Flows are usually defined as 5-tuple consisting of source and destination IP addresses, source and destination ports and protocol identifer. The low level analysis can generally catch a more finite characteristics of botnet related communication, while the packet level analysis can provide more information on the attack vectors as it inspects the packet payload. Additionally, as the low level analysis does not require access to the packet payload it is less privacy evasive comparing to the packet level analysis. Network-based detection can be classifed based on several aspects, such as the point of implementation, the stealthiness of operation, and the basic principles of functioning.

1.8.2 Client based detection method

Client-based detection methdot is one of the classic detection method. But it is also used nowadays. Client-based detection method needs port address

of the source and destination. It is very much useful while detecting peer to peer(p2p) based botnets.

1.9 Machine Learning for Botnet Detection

The basic assumption behind machine learning-based methods is that botnets produce distinguishable patterns of traf?c or behaviour within the client machine and that this patterns could be detected by employing some of the Machine Learning Algorithms (MLA).
Machine Learning (ML), is a branch of artificial intelligence, that has a goal of construction and studying of systems that can learn from data. Learning in this context implies ability to recognize complex patterns and make quali?ed decisions based on previously seen data. The main challenge of machine learning is how to provide generalization of knowledge derived from the limited set of previous experiences, in order to produce a useful decision for new, previously unseen, events. To tackle this problem the ?eld of Machine Learning develops an array of algorithms that discover knowledge from speci?c data and experience, based on sound statistical and computational principles. Machine learning relies on concepts and results drawn from many ?elds, including statistics, arti?cial intelligence, information theory philosophy, cognitive science, control theory and biology. The developed machine learning algorithms (MLAs) are at the basis of many applications, ranging from computer vision to language processing, forecasting, pattern recognition, games data mining, expert systems and robotics. At the same time important advances in the machine learning theory and algo rithms have promoted machine learning to the principal mean for discovering knowledge from the abundance of data that is currently available in diverse application areas. One of the emerging application areas is botnet detection that relies on MLAs to detect the bot-related network traf?c patterns. Machine learning algorithms can be classi?ed based on the desired outcome of the algorithm on two main classes:

1. Supervised Learning.
2. Unsupervised Learning.

Supervised learning is the class of well-defined machine learning algorithms that generate a function (i.e., model) that maps inputs to desired outputs. These algorithms aretrained by examples of inputs and their corresponding outputs, and then they are used to predict output for some future inputs. The

Supervised learning is used for classi?cation of input data on some de?ned class and for regression that predict continuous valued output.

Unsupervised learning is the class of machine learning algorithms where training data consists of a set of inputs without any corresponding target output values. The goal in unsupervised learning problems may be to discover groups of similar examples within the input data, where it is called clustering, to determine the distribution of data within the input space, known as density estimation, or to project the data from a high-dimensional space down to two or three dimensions for the purpose of visualization.

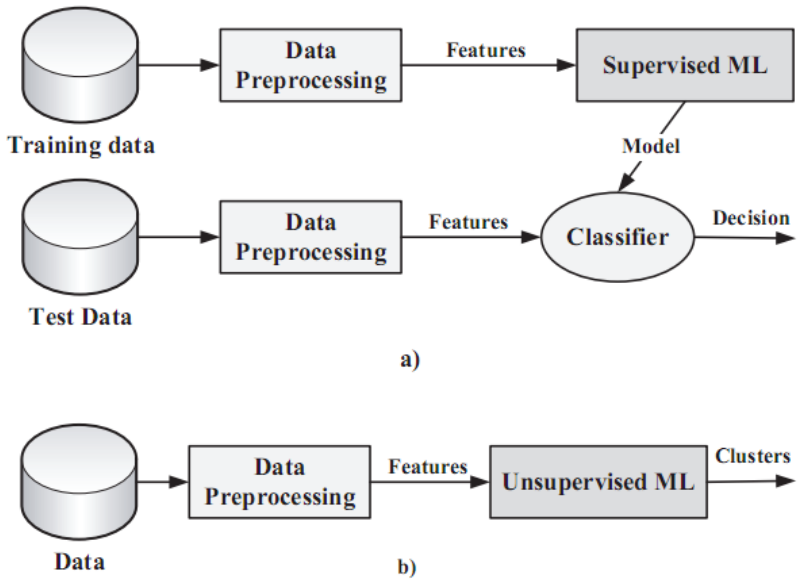

Figure 1.3: (a)supervised learning framework,(b)unsupervised learning framework

Chapter 2

Literature Survey

2.1 Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation

2.1.1 Abstract

They describe a simple, yet effective method to detect bot-infected machines within a given network that relies on detection of the communication channel between bot and Command & Control server (C&C server). The presented techniques are mainly based on passively monitoring network traffic for unusual or suspicious IRC nicknames, IRC servers, and uncommon server ports. By using n-gram analysis and a scoring system, they are able to detect bots that use uncommon communication channels, which are commonly not detected by classical intrusion detection systems. Upon detection, it is possible to determine the IP address of the C&C server, as well as, the channels a bot joined and the additional parameters which were set. The software Rishi implements the mentioned features and is able to automatically generate warning emails to report infected machines to an administrator.

Contribution

They introduce the background for their work on detecting infected machines. This method based on n-gram analysis and a scoring system to detect infected hosts within a given network.

System Architecture:

Project Rishi:

Rishi, their proof-of-concept implementation, monitors captured TCP packets for the occurrence of one of the following IRC commands: NICK, JOIN, USER, QUIT, and MODE. The parameters given with these commands are extracted and stored to be further analyzed by the program. The analysis focuses on the nick-name they extracted, all other parameters are just stored to collect additional information about the botnet, e.g., for tracking purposes. Every captured packet is then further analyzed by the script, which extracts the following information (if avail-able):

1. Time of suspicious connection.
2. IP address and port of suspected source host.
3. IP address and port of destination IRC server.
4. Channels joined.
5. Utilized nickname.

Scoring Function

After an appropriate packet has been captured and all necessary information has been extracted, the gathered nickname is passed to an analysis function. The analysis function implements a scoring function in order to estimate whether or not a given suspicious host is infected with a bot or not. The function checks for the occurrence of several criteria, like for example suspicious substrings, special characters, or long numbers. For each successful test, a certain number of points is added to the overall score the particular nickname has already received.

Regular Expression

Each nickname is tested against several regular expressions, which match known bot names. Currently, the configuration file contains 52 different regular expressions to match several hundred nicknames known to be used by bots. These regular expressions were generated by analyzing more than 4,000 different bots and their corresponding nicknames. To avoid false alarms, the

regular expression are very specialized, each matching only a few usernames. For example the following expression:

$$\backslash[[0-9]\backslash|[0-9]\{4,\}$$

matches nicknames like [0|1234] and another expression matches names like |1234. Although both regular expressions could be merged to one, they kept them separated to be more flexible with fine tuning. Another example is an expression like

$$\backslash[[0-9]\{1,2\}\backslash|[\mathrm{A}-\mathrm{Z}]\{2,3\}\backslash|[0-9]\{4,\}\backslash]\$$

which matches common bot nicknames like [00|DEU|597660], [03|USA|147700], or [0|KOR|43724] As all regular expressions are kept in a separate configuration file, existing ones can be easily adjusted or new ones can be added to keep up with the ever-increasing number of bots.

If a bot matches one of the regular expressions, the final score is raised by the minimum number of points needed to trigger an alarm. Thus, in their current configuration, another 10 points would be added.

Whitlisting

Rishi operates a dynamic whitelist, which automatically adds and removes nicknames ac-cording to their final score as returned by the analysis function. Each nickname, which receives zero points by the analysis function, is added to the dynamic whitelist. However, if the score of a whitelisted nickname raises above half of the points needed to trigger an alarm, it is removed from the list. If it exceeds the threshold of 10 points, the nickname is automatically added to the dynamic blacklist.

Blacklisting

Rishi also maintains a server blacklist for known C&C servers. If a connection to one of those servers is established, the final score is raised to the minimum number of points needed to trigger an alarm. Furthermore, if the destination port is not within the common list of IRC ports (currently 6666 and 6667), another point is added to the score.

2.2 BotSniffer: Detecting Botnet Command & Control Channels in Network Traffic

2.2.1 Abstract

They propose an approach that uses network-based anomaly detection to identify botnet C&C channels in a local area network without any prior knowledge of signatures or C&C server addresses. This detection approach can identify both the C&C servers and infected hosts in the network. Their approach is based on the observation that, because of the pre-programmed activities related to C&C, bots within the same botnet will likely demonstrate spatial-temporal correlation and similarity. For example, they engage in coordinated communication, propagation, and attack and fraudulent activities. Their prototype system, BotSniffer, can capture this spatial-temporal correlation in network traffic and utilize statistical algorithms to detect botnets with theoretical bounds on the false positive and false negative rates. They evaluated BotSniffer using many real-world network traces. The results show that BotSniffer can detect real-world botnets with high accuracy and has a very low false positive rate.

2.2.2 Contribution

Their research makes several contributions:

1. First, they study two typical styles of control used in centralized botnet C&C. The first is the "push" style, where commands are pushed or sent to bots. IRC-based C&C is an example of the push style. The second is the "pull" style, where commands are pulled or downloaded by bots. HTTP-based C&C is an example of the pull style. Observing the spatial-temporal correlation and similarity nature of these botnet C&Cs, they provide a set of heuristics that distinguish C&C traf?c from normal traffic.

2. Second, they propose anomaly-based detection algorithms to identify both IRC and HTTP based C&Cs in a port-independent manner. The advantages of their algorithms include: (1) they do not require prior knowledge of C&C servers or content signatures; (2) they are able to detect encrypted C&C, (3) they do not require a large number of bots to be present in the monitored network, and may even be able to detect a botnet with just a single member in the monitored network in some

cases, (4) they have bounded false positive and false negative rates, and do not require a large number of C&C communication packets.

3. Third, they develop a system, BotSniffer, which is based on their proposed anomaly detection algorithms and is implemented as several plugins for the open-source Snort. They have evaluated BotSniffer using real-world network traces. The results show that it has high accuracy in detecting botnet C&Cs with a very low false positive rate.

2.2.3 BotSniffer System Architecture:

There are two main components, i.e., the monitor engine and the correlation engine. The monitor engine is deployed at the perimeter of a monitored network. It examines network traffic, generates connection record of suspicious C&C protocols, and detects activity response behavior (e.g., scanning, spamming) and message response behavior (e.g., IRC PRIVMSG) in the monitored network. The events observed by the monitor engine are analyzed by the correlation engine. It performs group analysis of spatial-temporal correlation and similarity of activity or message response behaviors of the clients that connect to the same IRC or HTTP server. They implemented the monitor engines as several preprocessor plugins on top of the open-source system Snort, and implemented the correlation engine in Java. They also implemented a real-time message response correlation engine (in C), which can be integrated in the monitor engine. The monitor engines can be distributed on several networks, and collect information to a central repository to perform correlation analysis. They describe each BotSniffer component in the following sections.

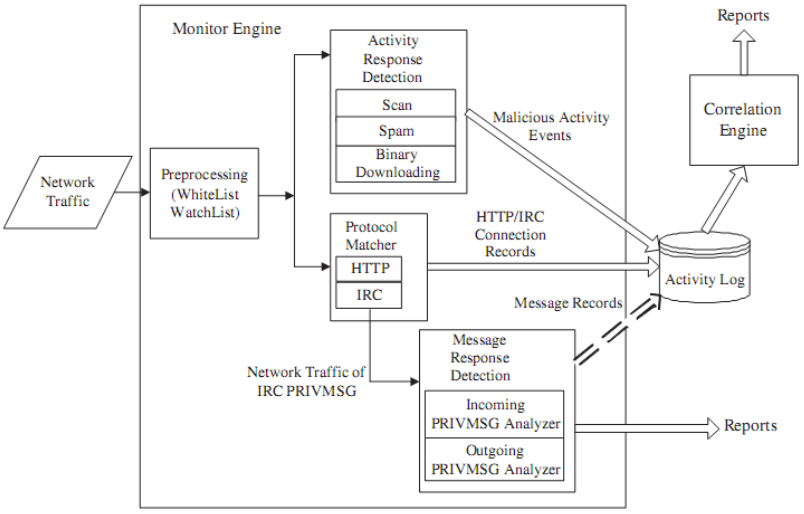

Figure 2.1: BotSniffer Architecture

2.2.4 Limitation

Evasion by misusing the whitelist. If a botmaster knows their hard whitelist, he may attempt to misuse these white addresses.

2.3 BotMiner: Clustering Analysis of Network Traffic for Protocol- and Structure-Independent Botnet Detection

2.3.1 Abstract

Botnets are now the key platform for many Internet attacks, such as spam, distributed denial-of-service (DDoS), identity theft, and phishing. Most of the current botnet detection approaches work only on specific botnet command and control (C&C) protocols (e.g., IRC) and structures (e.g., centralized), and can become ineffective as botnets change their C&C techniques. In this paper, they present a general detection framework that is independent of botnet C&C protocol and structure, and requires no a priori knowledge of botnets (such as captured bot binaries and hence the botnet signatures, and C&C server names/addresses). They start from the definition and es-

sential properties of botnets. They define a botnet as a coordinated group of malware instances that are controlled via C&C communication channels. The essential properties of a botnet are that the bots communicate with some C&C servers/peers, perform malicious activities, and do so in a similar or correlated way. Accordingly, their detection framework clusters similar communication traffic and similar malicious traffic, and performs cross cluster correlation to identify the hosts that share both similar communication patterns and similar malicious activity patterns. These hosts are thus bots in the monitored network. They have implemented their BotMiner prototype system and evaluated it using many real network traces. The results show that it can detect real-world botnets (IRC-based, HTTP-based, and P2P botnets including Nugache and Storm worm), and has a very low false positive rate.

2.3.2 Contribution

They develop a novel general botnet detection framework that is grounded on the definition and essential properties of botnets. Their detection framework is thus independent of botnet C&C protocol and structure, and requires no a priori knowledge (e.g., C&C addresses/signatures) of specific botnets. It can detect both centralized (e.g., IRC, HTTP) and current (and possibly future) P2P based botnets.
They define a new "aggregated communication flow (C-flow) record data structure to store aggregated traffic statistics, and design a new layered clustering scheme with a set of traffic features measured on the C-flow records. Their clustering scheme can accurately and ef?ciently group similar C&C traffic patterns.
They build a BotMiner prototype system based on their general detection framework, and evaluate it with multiple real-world network traces including normal traffic and several real-world botnet traces that contain IRC, HTTP and P2P-based botnet traffic (including Nugache and Storm). The results show that BotMiner has a high detection rate and a low false positive rate.

2.3.3 BotMiner Architecture

The architecture of BotMiner detection system, which consists of ?ve main components:

1. C-plane monitor

2. A-plane monitor
3. C-plane clustering module
4. A-plane clustering module
5. Cross-plane correlator.

The two traffic monitors in C-plane and A-plane can be deployed at the edge of the network examining traffic between internal and external networks. They run in parallel and monitor the network traffic. The C-plane monitor is responsible for logging network flows in a format suitable for efficient storage and further analysis, and the A-plane monitor is responsible for detecting suspicious activities (e.g., scanning, spamming, and exploit attempts). The C-plane clustering and A-plane clustering components process the logs generated by the C-plane and A-plane monitors, respectively. Both modules extract a number of features from the raw logs and apply clustering algorithms in order to find groups of machines that show very similar communication (in the C-plane) and activity (in the A-plane) patterns. Finally, the cross-plane correlator combines the results of the C-plane and A-plane clustering and makes a final decision on which machines are possibly members of a botnet. In an ideal situation, the traffic monitors should be distributed on the Internet, and the monitor logs are reported to a central repository for clustering and cross-plane analysis.

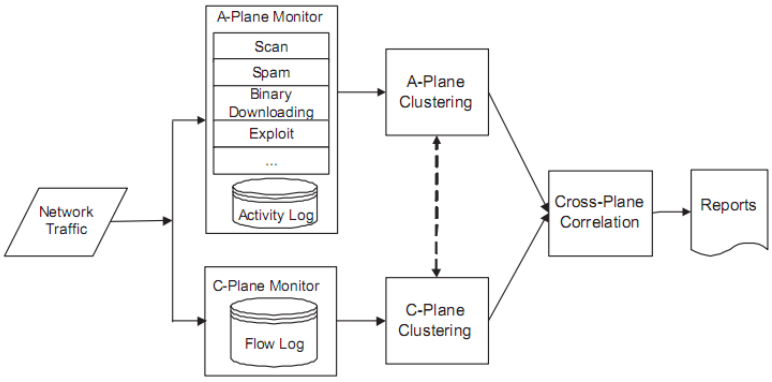

Figure 2.2: BotMiner Architecture

2.3.4 Limitations and Potential Solutions

Like any intrusion/anomaly detection system, BotMiner is not perfect or complete. It is likely that once adversaries know their detection framework and implementation, they might ?nd some ways to evade detection, e.g., by evading the C-plane and A-plane monitoring and clustering, or the cross-plane correlation analysis. They now address these limitations and discuss possible solutions.

2.4 JACKSTRAWS: Picking Command and Control Connections from Bot Traffic

2.4.1 Abstract

A distinguishing characteristic of bots is their ability to establish a command and control (C&C) channel. The typical approach to build detection models for C&C traffic and to identify C&C endpoints (IP addresses and domains of C&C servers) is to execute a bot in a controlled environment and monitor its outgoing network connections. Using the bot traffic, one can then craft signatures that match C&C connections or blacklist the IP addresses or domains that the packets are sent to. Unfortunately, this process is not as easy as it seems. For example, bots often open a large number of additional connections to legitimate sites (to perform click fraud or query for the current time), and bots can deliberately produce "noise" - bogus connections that make the analysis more dif?cult. Thus, before one can build a model for C&C traffic or blacklist IP addresses and domains, one first has to pick the C&C connections among all the network traffic that a bot produces.

In this paper, they present JACKSTRAWS, a system that accurately identifies C&C connections. To this end, they leverage host-based information that provides insights into which data is sent over each network connection as well as the ways in which a bot processes the information that it receives. More precisely, they associate with each network connection a behavior graph that captures the system calls that lead to this connection, as well as the system calls that operate on data that is returned. By using machine learning techniques and a training set of graphs that are associated with known C&C connections, they automatically extract and generalize graph templates that capture the core of different types of C&C activity. Later, they use these C&C templates to match against behavior graphs produced by other bots.

Their results show that JACKSTRAWS can accurately detect C&C connections, even for novel bot families that were not used for template generation.

2.4.2 Contribution

The contributions of this paper are the following:

1. They present a novel approach to identify C&C communication in the large pool of network connections that modern bots open. Their approach leverages host-based information and associates models, which are based on system call graphs, with the data that is exchanged over network connections.

2. They present a novel technique that generalizes system call graphs to capture the "essence" of, or the core activities related to, C&C communication. This generalization step extends previous work on system call graphs, and provides interesting insights into the purpose of C&C traffic.

3. They implemented these techniques in a tool called JACKSTRAWS and evaluated it on 130,635 connections produced by more than 37 thousands malware samples. Their results show that the generated templates detect known C&C traffic with high accuracy, and less than 0.2% false positives over harmless traffic. Moreover, they found 9,464 previously unknown C&C connections, improving the coverage of handcrafted network signatures by 60%.

2.4.3 System Model Architecture

They provide an overview of the actual implementation of JACKSTRAWS and explain the different analysis steps in greater details.

Environment Analysis

They use the dynamic malware analysis environment Anubis as the basis for their implementation, and implemented several extensions according to their needs. Note that the general approach and the concepts outlined in this paper are independent of the actual analysis environment; they could have also used BitBlaze, Ether, orany other dynamic malware analysis environment.

Behavior Graph Generation

When the sample and all of its child processes have terminated, or after a fixed timeout (currently set to 4 minutes), JACKSTRAWS saves all monitored system calls, network-related data, and tainting information into a log file. Unlike previous work that used behavior graphs for distinguishing between malicious and legitimate programs, they use these graphs to determine the purpose of network connections (and to detect C&C traffic). Thus, they are not interested in the entire activity of the malware program. Instead, they only focus on actions related to network traffic. Next they Label the graph and Simplify the behavior graph.

Graph Mining

The first step when generating C&C templates is graph mining. More precisely, the goal is to mine frequent sub-graphs that are only present in the malicious set. They process graphs through following:

- Frequent subgraph mining.
- Subgraph maximization
- Graph sets difference

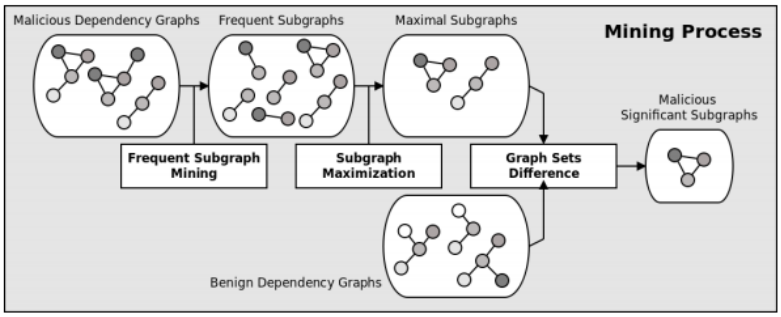

Figure 2.3: Mining Process

Graph Clustering

Using as input the frequent, malicious subgraphs produced by the previous mining step, the purpose of this step is to find clusters of similar graphs. The

graph mining step produces different graphs that represent different types of behaviors. They now need to cluster these graphs to find groups, where each group shares a common core of activities (system calls) typical of a particular behavior. Graph clustering is used for this purpose; generated clusters are later used to create generalized templates covering the graphs they contain.

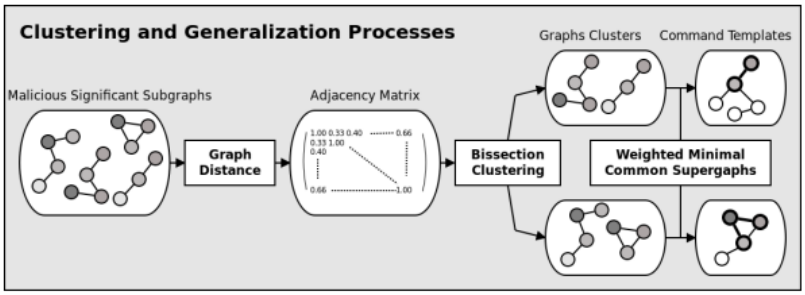

Figure 2.4: Clustering and Generalization Process

A crucial component for every clustering algorithm is the proper choice of a similarity measure that computes the distances between graphs. In their system, the similarity measure between two graphs is based on their non-induced, maximum common subgraph (mcs). The similarity measure is defined as:

$$d(G1, G2) = 2 * |edges(mcs(G1, G2))|/|edges(G1)| + |edges(G2)|$$

Graph Generalization and Templating

Based on the clusters of similar graphs, the final step in their template generation process is graph generalization. The goal of the generalization process is to construct a template graph that abstracts from the individual graphs within a cluster. Intuitively, they would expect that a template contains a core of nodes, which are common to all graphs in a cluster. In addition, to capture small differences between these graphs, there will be optional nodes attached to the core. The generalization algorithm computes the weighted minimal common supergraph (WMCS) of all the graphs within a given cluster.

Template Matching

The previous steps leveraged machine learning techniques and sets of known malicious and benign graphs to produce a number of C&C templates. These templates are graphs that represent host-based activity that is related to command and control traffic. To find the C&C connections for a new malware sample, this sample is first executed in the sandbox, and their system extracts its behavior graphs. Then, they match all C&C templates against the behavior graphs. Whenever a match is found, the corresponding connection is detected as command and control traffic, and they can extract its endpoints (IPs and domains)

Limitations

They aim at analyzing malicious software, which is a hard task in itself. An attacker can use different techniques to Interfere with the analysis environment which is of concern for them, such as unnecessary system calls, nodes etc.

2.5 BotFinder: Finding Bots in Network Traffic without Deep Packet Inspection

2.5.1 Abstract

Present BOTFINDER, a novel system that detects infected hosts in a network using only high-level properties of the bot's network traffic. BOTFINDER does not rely on content analysis. Instead, it uses machine learning to identify the key features of command-and-control communication, based on observing traffic that bots produce in a controlled environment. Using these features, BOTFINDER creates models that can be deployed at network egress points to identify infected hosts.

2.5.2 Contribution

This paper makes the following contributions:

1. They observe that C&C traffic of different bot families exhibits regularities (both in terms of traf?c properties and timing) that can be

leveraged for network-based detection of bot-infected hosts. Being independent of packet payloads, their detection approach can handle encrypted or obfuscated traffic.

2. They present BOTFINDER, a learning-based approach that automatically generates bot detection models. To this end, they run bot binaries in a controlled environment and record their traffic. Using this data, they build models of characteristic network traffic features.

3. They develop a prototype of BOTFINDER, and they show that the system is able to operate on high-performance networks with hundreds of thousands of active hosts and Gigabit throughput in real time. They apply BOTFINDER to real traffic traces and demonstrate its high detection rate and low false positive rate. Additionally, they show that BOTFINDER outperforms existing bot detection systems and discuss how BOTFINDER handles certain evasion strategies by adaptive attackers.

2.5.3 System Architecture

BOTFINDER detects malware infections in network traffic by comparing statistical features of the traffic to previously-observed bot activity. Therefore, BOTFINDER operates in two phases: a training phase and a detection phase. During the training phase, their system learns the statistical properties that are characteristic of the command and control traffic of different bot families. Then, BOTFINDER uses these statistical properties to create models that can identify similar traffic. In the detection phase, the models are applied to the traffic under investigation. This allows BOTFINDER to identify potential bot infections in the network, even when the bots use encrypted C&C communication.

Input Data Processing

The input for BOTFINDER is either a traffic capture or NetFlow data. During the training phase, malware samples are executed in a controlled environment, and all network traffic is recorded. In this step, it is important to correctly classify the malware samples so that different samples of the same malware family are analyzed together.

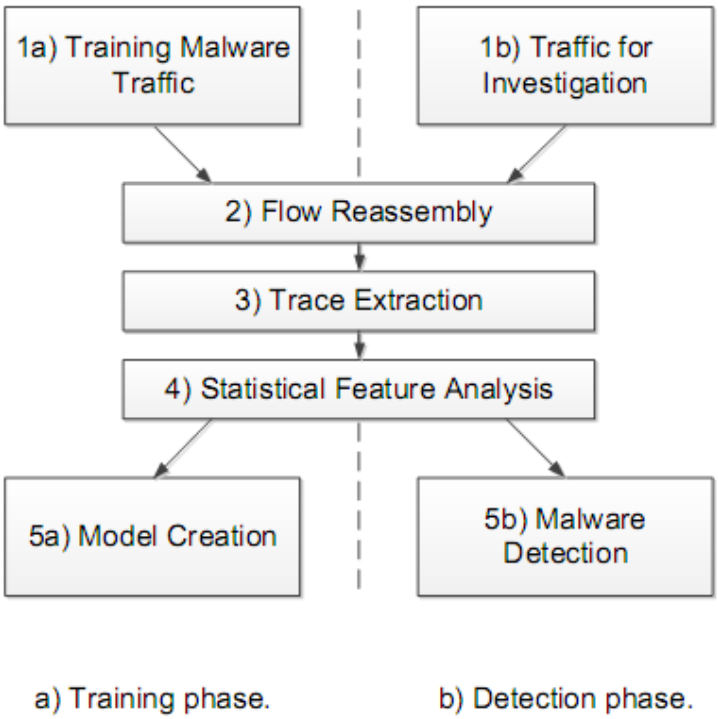

Figure 2.5: General Architecture of BotFinder

Flow Reassembly

If NetFlow data is available, BOTFINDER directly imports it; otherwise, they reassemble flows from captured packet data. For each connection properties such as start and end times, the number of bytes transferred in total, and the number of packets is extracted. As a final result of this reassembly step, their system yields aggregated data similar to NetFlow, which is the industry standard for traffic monitoring and IP traffic collection. For all further processing steps, BOTFINDER only operates on these aggregated, content-agnostic data.

Trace Extraction

Traces are an important concept in BOTFINDER: A trace T is a sequence of chronologically-ordered flows between two network endpoints. To obtain meaningful statistical data, they require at least a minimal number of connections j Tj min (typically between 10 and 50) in a trace T. This minimal length requirement is consistent with the fact that command-and-control traffic consists of multiple connection.

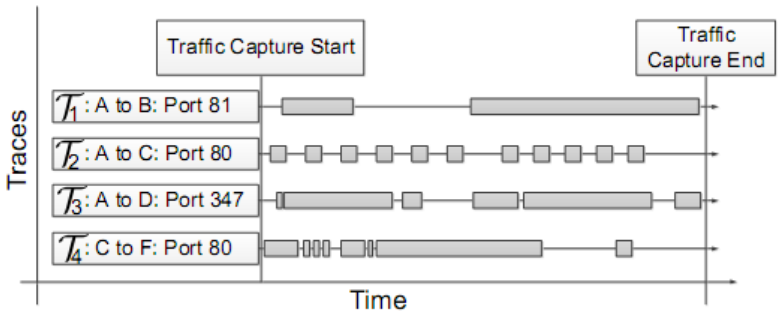

Figure 2.6: Traces with different statistical behaviour

Feature Extraction

After trace generation, BOTFINDER processes each trace to extract relevant statistical features for subsequent trace classification. They focus on the following 5 features:

1. The average time interval between the start times of two sub sequent flows in the trace.
2. The average duration of connections.
3. The average number of source bytes per flow.
4. The average number of destination bytes per flow.
5. The Fast Fourier Transformation (FFT).

Model Creation

They create models via clustering of the 5 features: average time, average duration, average source bytes, average destination bytes, and the FFT. They process the dataset for each feature separately, as they observed malware behavior with non-correlated features. As an example of such behavior, two versions of a bot might connect to different versions of C&C servers C1 and C2 and transfer different amounts of bytes, depending of their version. Nevertheless, these two bot versions might still follow the same communication periodicity pattern.

Model Matching(Detection)

Now in the detection face they check the botnet traffic with the model clusters. If the clusters model matches then it detects and raises an alarm.

2.6 BotMosaic: Collaborative Watermark for Botnet Traceback

2.6.1 Abstract

They study the problem of finding a botmaster who is controlling a botnet. Botmasters will typically use several stepping stones to connect to the botnet to disguise their true location. While there has been a lot of work on detecting stepping stones, it has focused on interactive stepping stones that follow a different pattern of communication than botnets. Thus a different approach is necessary. They develop a new network flow watermarking scheme, called BotMosaic. The scheme works by capturing multiple bots within a honeynet

and then inserting a watermark onto the communication stream of each bot towards the botmaster. By creating a collaborative watermark that is spread across several bot communication streams, it is possible to overcome the particular nature of botnet command-and-control communication that renders other watermarking techniques ineffective. Given a sufficient collection of trapped bots, the botmaster's connection can be detected with very low false error rates. BotMosaic detection is low-cost so it can be easily deployed by ISPs and enterprises to help address the botnet problem. It can also be used to locate internal hosts that are part of the botnet, i.e., bots, and also internal compromised machines being used as botnet stepping stones, providing an additional incentive for incremental deployment.

2.6.2 Contribution

Their approach is to capture a collection of bots inside a honeynet. They then insert a collaborative watermark that is spread across the communication stream of the entire collection of captured bots. Each stream is minimally modi?ed, but as the streams are aggregated on the way to the botmaster, the combined watermark becomes visible. The watermark can be detected at a very low cost by detectors deployed at ISPs and enterprise border routers. Their watermark can also be used to locate stepping stones-compromised computers used by botmasters-as well as infected bot machines within a network. As such, there is added incentive for incremental deployment.

They analyze their scheme using simulations and experiments on PlanetLab. They find that they can achieve a high rate of detection with few false positives using a watermark applied to small number of captured/imitated bots in the network, and the resulting detection time of about a minute.

2.6.3 System Architecture

Their aim is to design a watermark-based approach for the botnet trace back problems in an IRC-based botnet, i.e., tracing back a botmaster, detecting bots and botnet-related stepping stones. The work is different from similar watermarks in being collaborative by using multiple rogue bots for watermark insertion, which makes it specialized for the problem of botnet watermarking. Rogue bots are the bots collected/imitated and controlled by the watermarking system, as opposed to real bots. Multiple rogue bots allow them to spread the watermark power over a larger number of communica-

tions, compensating for the small amount of communication each individual bot performs with the botmaster. BotMosaic adopts an intervalbased design used in other watermarks to provide robustness to packet losses, delays, and reordering as well as chaff introduced by the traf?c of the other bots in the botnet. To obtain a large number of rogue bots, they propose using a darknet operation together with a honeynet framework. A network telescope can be used to route a large segment of the dark address space to a cluster of honeypot machines. The machines themselves can use virtualization to appear as a large number of hosts with various system con?gurations. Over time, they can expect that a number of the virtual honeypot machines will be compromised and start hosting bots that participate in a botnet. It may also be possible to deliberately infect other machines to increase the number of rogue bots. They then insert a watermark in a coordinated fashion into the communications of all of the rogue bots. This allows them to insert a much stronger signal into the encrypted communication link back towards the botmaster/bots that is detectable even if these communications are mixed with traffic from other, real bots. The last component of their architecture is a series of detectors deployed around the Internet. The detectors maintain loose time synchronization with the watermarkers and use interval-based detection to account for packet delays, drops, or reordering.

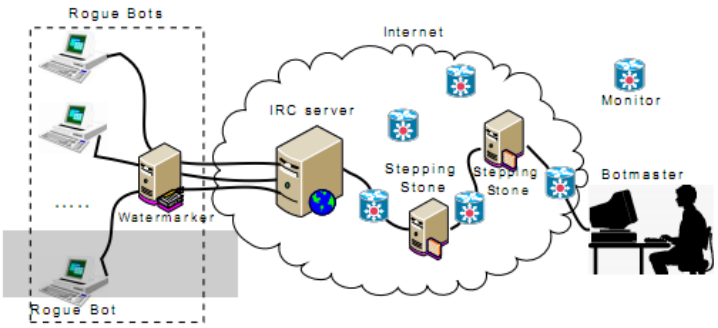

Figure 2.7: Botmosaic watermarking traceback scheme

2.7 BotHunter: Detecting Malware Infection through IDS-Driven Dialog Correlation

2.7.1 Abstract

They present a new kind of network perimeter monitoring strategy, which focuses on recognizing the infection and coordination dialog that occurs during a successful mal-ware infection. BotHunter is an application designed to track the two-way communication flows between inter-nal assets and external entities, developing an evidence trail of data exchanges that match a state-based infection sequence model. BotHunter consists of a correlation engine that is driven by three malware-focused network packet sensors, each charged with detecting specific stages of the malware infection process, including inbound scanning, exploit usage, egg downloading, outbound bot coordination dialog, and outbound attack propagation. The BotHunter correlator then ties together the dialog trail of inbound intrusion alarms with those outbound communication patterns that are highly indicative of successful local host infection. When a sequence of evidence is found to match BotHunter's infection di-alog model, a consolidated report is produced to capture all the relevant events and event sources that played a role during the infection process. We refer to this analytical strategy of matching the dialog flows between internal assets and the broader Internet as dialogbased correlation, and contrast this strategy to other intrusion detection and alert correlation methods. We present our experimental results using BotHunter in both virtual and live testing environments, and discuss our Internet release of the BotHunter prototype. BotHunter is made available both for operational use and to help stimulate research in understanding the life cycle of malware infections.

2.7.2 System Architecture

BotHunter System Architecture is given below:

Now they turn their attention to the design of a passive monitoring system capable of recognizing the bidirectional warning signs of local host infections, and correlating this evidence against our dialog infection model. Our system, referred to as BotHunter, is composed of a trio of IDS components that monitor in- and out-bound traffic flows, coupled with our dialog correlation engine that produces consolidated pictures of successful bot infections. We envision BotHunter to be located at the boundary of a network, providing

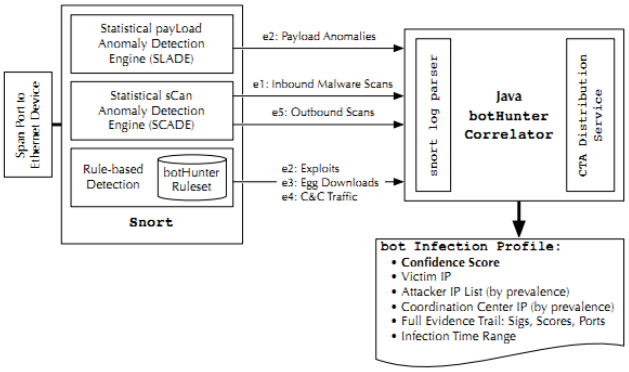

Figure 2.8: BotHunter System Architecture

it a vantage point to observe the network communication flows that occur between the network's internal hosts and the Internet. Their IDS detection capabilities are composed on top of the open source release of Snort. We take full advantage of Snort's signature engine, incorporating an extensive set of malware-specific signatures that we developed internally or compiled from the highly active Snort community. The signature engine enables us to produce dialog warnings for inbound exploit usage, egg downloading, and C&C patterns. In addition, they have developed two custom plugins that complement the Snort signature engine's ability to produce certain dialog warnings. Note that they refer to the various IDS alarms as dialog warnings because we do not intend the individual alerts to be processed by administrators in search of bot or worm activity. Rather, they use the alerts produced by our sensors as input to drive a bot dialog correlation analysis, the results of which are intended to capture and report the actors and evidence trail of a complete bot infection sequence

Chapter 3

Motivation

3.1 Motivation

In this chapter we discuss about our motivation. Today's life is very much depending upon network and internet. Our many important documents we keep in our computerized system. Malicious attackers try to hack our system to get our valuable information. Most of the organizations are now depending upon computing based system. Their works and all the documents from the past till present time are kept in a computerized system. So the security of that system is very much important.

Network is a very available medium for data communication. For transferring data, file sharing, voice mail, video conference all are network communication. Malicious attackers find out many systems for attack the network system for malicious reasons. The most efficient, and arguably, most relevant kind of such malware are bots. The malicious software components are coordinated over a command and control (C&C) channel by a single entity called the botmaster and form a botnet to carry out a number of different criminal activities. Consequently, defenses against malware infections are a high priority, and the identification of infected machines is the first step to purge the Internet of bots.

Many security measures have been invented to detect the botnet. According to the nature of botnet the researchers find out Different detection approaches. For detecting the botnet they proposed many algorithm and take different features. But none of them have mentioned any feasibility of minimal sufficient features.

So we try to find out the most feasible features. We did a serve and found 14 unique feature lists from 30 research papers. We will find out the most minimal sufficient features among them by appropriate mechanism.

Again among all the research papers most of the approach is for detecting the infected host and the botnet.BotMosaic is about finding BotMaster but it use honetnet and watermarking approach which is not good enough to detect botmaster as well as botnet. They need 10-15% rough bot for effectiveness. They also need to establish the system before attack occurs.

There are many models using machine learning approach and anomaly based detection method, which main concern is about botnet detection. Although BotMosaic is a very primary approach for BotMaster detection it has many defects.

There is no effective approach for detecting botnet as well as botmaster. If we only concern about botnet then the botmaster can again attack the system or another. So we need to find out the botnet as well as the botmaster.

Chapter 4

Our Proposal

4.1 Proposed Model

Our main proposal is to make such a model which can detect the botnet as well as the BotMaster. Our proposed model has 4 phases:

1. Feature Ranking
2. Training Phase
3. Botnet Detection Phase
4. BotMaster Detection Phase

We found a list of unique feature. Ranking them we will find out the best minimal sufficient features. The list is given below:

- Flow Duration
- Average time interval between the start time of two subsequence
- Average no of source and destination bytes
- Average packet length
- Maximum initial congestion window
- Flow protocol
- Time of packet arrival

- Percentage of packets pushed in flow
- Payload size
- Traffic frequency
- Ratio of incoming packets over outgoing packets
- Number of connections
- Average bits per second for flow

For the training phase and botnet detecting face we will use effective machine learning algorithm based on our features.

The last phase is for detecting the Botmaster. For this we will use a IP traceback approach.

4.2 IP Traceback

The goal of IP Traceback is to trace the path of an IP packet to its origin. Two main kinds of IP traceback techniques:

1. Packet marking
2. Packet logging

4.3 Our System Architecture

Our Proposed System Model is given below:

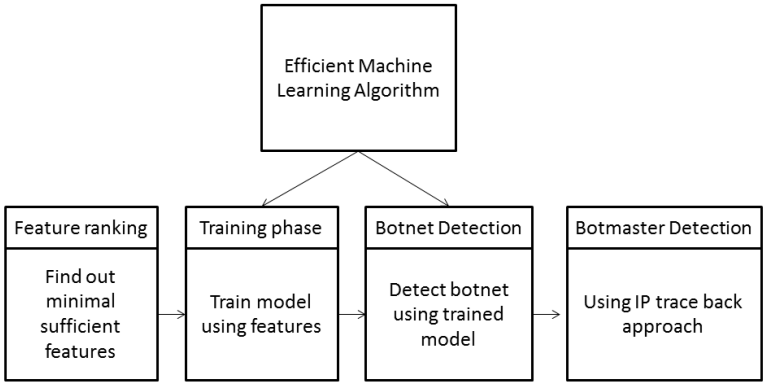

Figure 4.1: BotEliminator System Architecture

4.3.1 Feature Ranking Phase

4.3.2 Machine Learning for Training Model Phase

Train Model

Botnet Detection

4.3.3 Botmaster Detection Phase

4.4 IP Traceback for Botmaster Detection

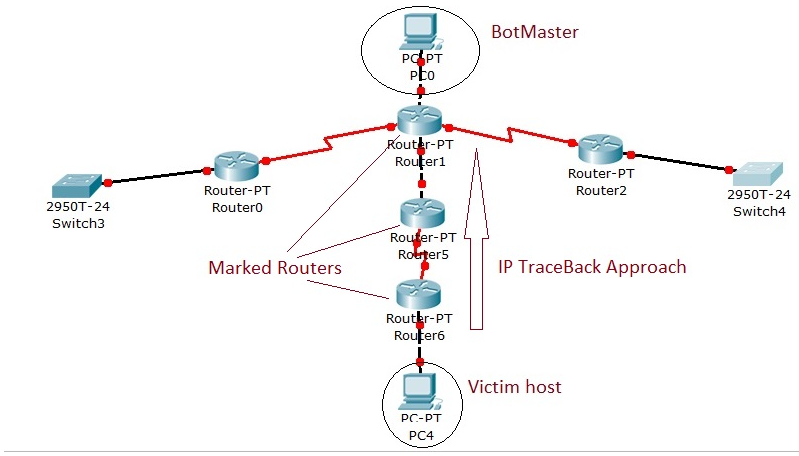

Figure 4.2: Botmaster Detection using IP Traceback Approach

Chapter 5

Problems Encountered

5.1 Problems we Faced

- We found some existing codes for detecting Botnet, but we failed to compile them properly.
- We try to detect botmaster for IRC based server as well as HTTP based server.
- It is quiet impossible for us to detect HTTP based server because we need many bot at a time.
- Botmaster keep their identity secret and also some cases they connect with the C&C server for a short period of time.
- Most of the works we have study so far is not completely detect the botnet as well as the botmaster and also bot the IRC based and HTTP based server.
- Hybrid configuration for the C&C server is now becoming very dangerous.
- There are too many methods used by the botmaster to make them secret.

Chapter 6

Scopes of Work

6.1 Scopes of Work

- In future there is a scope for detecting the whole bot network including Botmaster
- Hybrid botnet is becoming more and more dangerous so it need to be solved
- IP trace back approach is currently used for IRC based model but not for HTTP based model
- The features used for detecting botnet need to be summarized

Chapter 7

Conclusion

Botnet detection is a relatively new and a very challenging research area. Botnet detection based on machine learning has been the subject of interest of the research community resulting in the numerous detection methods that are based on different botnet heuristics, that target different types of botnets using diverse machine learning algorithms and that consequently provide varying performances of detection. We presented BotEliminator, a novel malware detection system that is based on statistical network flow analysis. After a fully-automated, non supervised training phase on known bot families, BotEliminator creates models based on the clustered statistical features of command and control communication. BotEliminator is able to operate on fully-encrypted traffic, raising the bar for malware authors. BotEliminator is also the first example of finding Botmaster with accurate botnet detection. We hope that our Internet release will enable the community to extend and maintain this capability while inspiring new research directions. Our ongoing work involves improving the detection accuracy of BotEliminator and its resilience to evasion, and performing more evaluation and deploying BotSniffer in the real world. We are also developing a next generation detection system that is independent of the protocol and network structure used for botnet Command and Control channel. We want to make sure the perfect and minimal features for detecting botnet using machine learning technique and a very easy and appropriate approach for detecting botmaster. BotEliminator is not an established approach but we hope in future it will add a very important role for finding botnet as well as botmaster.

Chapter 8

References

1 J. Goebel and T. Holz. Rishi: Identify bot contaminated hosts by irc nickname evaluation. In USENIX Workshop on Hot Topics in Understanding Botnets (HotBots'07), 2007.

2 G. Gu, J. Zhang, and W. Lee. BotSniffer: Detecting botnet command and control channels in network traf?c. In Proceedings of the 15th Annual Network and Distributed System Security Symposium (NDSS'08), 2008

3 G. Gu, R. Perdisci, J. Zhang, and W. Lee. BotMiner: Clustering Analysis of Network Traf?c for Protocol- and Structure-Independent Botnet Detection. In USENIX Security, 2008.

4 G. Jacob, R. Hund, C. Kruegel, and T. Holz. Jackstraws: Picking Command and Control Connections from Bot Traffic. USENIX Security, 2011.

5 Florian Tegeler, Xiaoming Fu, Giovanni Vigna, Christopher Kruegel. BotFinder: Finding Bots in Network Traf?c Without Deep Packet Inspection. Co-NEXT 2012

6 G. Gu, P. Porras, V. Yegneswaran, M. Fong, and W. Lee.BotHunter: Detecting Malware Infection Through IDS-Driven Dialog Correlation. In USENIX Security, 2007.

7 N. Kiyavash, A. Houmansadr, and N. Borisov, "Multiflow attacks against network flow watermarking schemes," in USENIX Security Symposium, P. van Oorschot, Ed. Berkeley, CA, USA: USENIX Association, 2008.

8 MMatija Stevanovic,Jens Myrup Pedersen. Machine learning for identifying botnet network traffic. Technical report.

9 Amir Houmansadr, Nikita Borisov. BotMosaic: Collaborative Watermark for Botnet Traceback. Arxiv 2012