DETECTION OF BOTNETS MOUNTED ON THE SESSION INITIATION PROTOCOL

by

Mohammad AlKurbi

B.Sc., College of Computer and Information Sciences, King Saud University, Riyadh, Saudi Arabia, 1998

A Project submitted in partial fulfillment
of the requirements for the degree of
Master of Science
in the School
of
Computing Science

SIMON FRASER UNIVERSITY
Spring 2011

APPROVAL

Name: Mohammad AlKurbi

Degree: Master of Science

Title of Project: Detection of botnets mounted on the Session Initiation Protocol

Examining Committee: Dr. Kay C. Wiese
Chair

Dr. Mohamed Hefeeda, Senior Supervisor

Dr. Robert D. Cameron, Supervisor

Dr. Joseph G. Peters, Examiner

Date Approved: 30 March 2011

Declaration of
Partial Copyright Licence

Abstract

A botnet is a group of compromised computers (called bots) controlled by remote attackers to distribute spam emails, launch denial of service attacks, and perform other malicious activities. Botnets can be deployed on top of different protocols, such as the Internet Relay Chat (IRC), the Hyper Text Transfer Protocol (HTTP), and the Session Initiation Protocol (SIP). The SIP is widely used to initiate voice over IP, and it has been recently adopted by the telecommunications standards bodies to be the signaling protocol for mobile telecommunication core networks. Such adoption will introduce a huge number of potential devices to botnets. Therefore, botnets deployed over the SIP present a serious threat for the Internet. We propose a novel approach to detect SIP botnets by looking for users who behave in similar and coordinated patterns. We show through extensive experimental evaluations that the proposed approach achieves low false positive and false negative rates.

Keywords: Botnet; Session Initiation Protocol; Botnet Detection Algorithms

Contents

List of Tables

List of Figures

Chapter 1

Introduction

In this chapter, we provide a brief introduction to botnets, state the addressed problem in this project, and summarize the project contributions.

1.1 What is a Botnet?

A botnet is a group of compromised computers running malicious software. The compromised computers are called bots. Bots are controlled by remote attackers, which are called botmasters. A botmaster controls bots through a Command and Control channel to organize different kinds of malicious activities. There are two types of the malicious activities: local and external. The local malicious activities include keyboard logging and password cracking [1]. The external malicious activities are the most common malicious activities for botnets, which include distributed denial of service attacks, spamming, and phishing [2, 3].

The Command and Control channel can be deployed over different protocols such as the Internet Relay Chat, the Hyper Text Transfer Protocol, the Peer to Peer (P2P) protocols, and the Session Initiation Protocol. Therefore, as shown in Figure 1.1, a botnet is composed of a botmaster who is a remote attacker controlling the botnet, and a Command and Control channel used to pass botmaster instructions to bots. The botmaster instructions are hidden within a legitimate traffic such as chat, web, voice over IP, and P2P messages.

A bot life cycle consists mainly of the following phases [1, 4]. The first is the Infection phase, which happens when a victim computer gets infected by an email attachment, accesses a compromised web site, or downloads a malicious software that exploits some vulnerabilities in the victim's operating system. The second is the bootstrap phase, where the

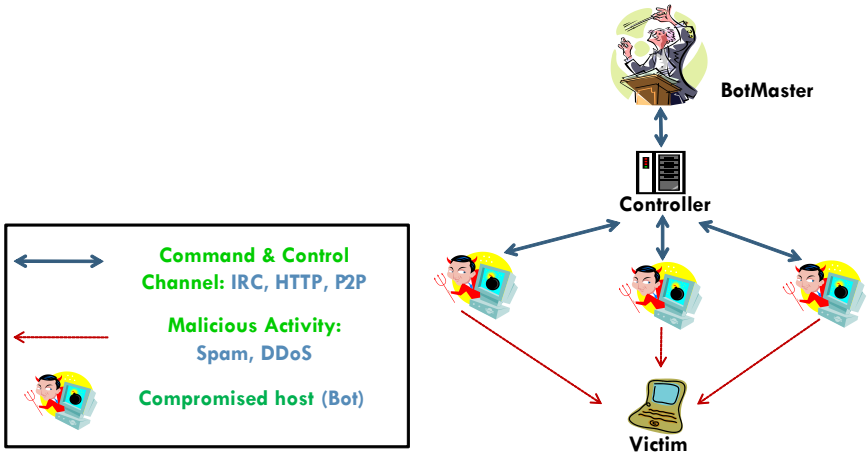


Figure 1.1: Botnet components

new infected computer finds and joins the botnet via initial list of key peers. The third is the Command and Control phase, where bots are ready to receive botmaster instructions through the Command and Control channel. The fourth is the propagation phase, which can be implemented in two ways. The first is the passive way, where a bot injects itself in the file sharing directory to infect any bots downloading those malicious files. The second is the active way, where a bot picks candidates from a candidate list, and tries to exploit operating system vulnerabilities in order to upload and install the bot software on them. The fifth is the malicious activities phase. The last phase is the maintenance phase, where bots patch their software through the Command and Control channel to fix a bug or add new features.

1.2 Problem Statement

Botnets are among the most serious and growing threats on the Internet [5–7]. In addition to the 40% of Internet computers connected to Internet that are considered potential bots in a recent study [4], SIP extends the problem domain by introducing new potential bots to it. SIP is widely used to initiate voice over IP sessions and it has been recently adopted by the telecommunications standards bodies, i.e., 3GPP [8], to be the standard signaling protocol for mobile telecommunication core networks. Such adoption will introduce a huge number of potential devices to botnets. The problem addressed in this project is to design a new approach to detect botnets formed on top of computing devices that utilize SIP as the

signaling protocol. Computing devices that use SIP include personal computers, netbooks, smart phones, and personal digital assistants (PDAs). The algorithm should detect botnets efficiently and in a timely manner.

1.3 Project Contributions

The contributions of this project can be summarized as follows:

- We propose a novel approach to detect SIP botnets by looking for users who behave in similar and coordinated patterns.
- We design and implement an online detection system based on the proposed approach.
- We conduct extensive experimental evaluations to test the system precision. Our results indicate that the proposed approach achieves low false positive and false negative rates.

1.4 Project Organization

The rest of the project is organized as follows. In Chapter 2, we provide an overview on botnet structures and the Session Initiation Protocol and how it can be a useful Command and Control protocol for botnets. We also summarize related works in the literature in this chapter. In Chapter 3, we present the proposed solution. In Chapter 4, we present the experimental evaluation of the proposed solutions. In Chapter 5, we conclude the project and outline potential extensions for this work.

Chapter 2

Background and Related work

2.1 Botnet Structures

A botnet can be deployed over different protocols such as the Internet Relay Chat, the Hyper Text Transfer Protocol, the P2P protocols, and the Session Initiation Protocol. The botnet structure is defined based on the Command and Control channel model which comes mainly in two forms: centralized and distributed. Internet Relay Chat and Napster botnets are example of centralized structure, where botmaster communicates with bots through central controllers, e.g., IRC servers, Figure 1.1. A centralized botnet is easy to manage, because message delivery is fast and can be guaranteed. However, a centralized botnet is easy to break [5]. In a distributed structure, such as a P2P botnet, Figure 2.1, controllers are scattered through out the network, which requires more management overhead, slower or sometimes unguaranteed message delivery. However, a distributed botnet has more resistance for mitigation techniques according to the dynamic and redundant set of controllers [1, 9]. A distributed botnet also known as a decentralized botnet.

To clarify a P2P botnet functionality, it is important to explain how the Command and Control channel works. There are two main techniques: Push and Pull [1,6]. The push style is a slow mechanism, where the botmaster pushes the commands to bots through controllers. It is similar to the Internet Relay Chat mechanism. The push mechanism in P2P network is done by pushing the commands to selected peers, and they pass them further to another selected list of peers, based on specific criteria [1], and so on. In the pull mechanism, also called publish/subscribe mechanism, controllers publish predetermined set of keywords, and bots query those controllers about those keywords to receive new instructions. Bots know

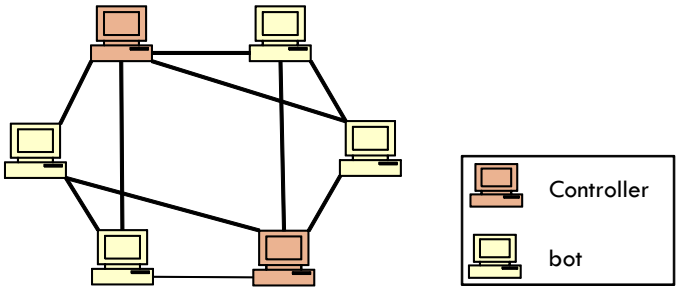


Figure 2.1: Distributed structure such as P2P botnet

where to send the query by looking up a distributed hash table, which works as a command index.

Dagon et al. [10] identify three important botnet features: effectiveness, efficiency, and robustness. Botnet potential damage, i.e., effectiveness, can be measured by the size of the largest connected bots that participate in the Command and Control channel, and the total bandwidth that botmaster can generate from the bots. The second feature is the communication efficiency which can be measured by the botnet diameter. When the diameter is large, the communication becomes slower and the detection possibility becomes higher as message passes many network nodes. The last feature is the robustness which indicates how a botnet can resist a disruption. It can be measured by the redundancy/replication of servers and links between bots. In order to avoid detection, a botmaster seeks a Command and Control channel with the following features [11]: low management overhead, looks legitimate, quiet, and efficient.

To mitigate the botnet power, bot communications have to be disrupted. Different botnet topologies require different response strategies. Cooke et al. [12] recommend mitigating the whole botnet by taking down the key nodes, such as botnet controllers, instead of taking down individual bots. Dagon et al. [10] confirm that the weakest link of a botnet is the Command and Control channel and targeting high degree nodes increases botnet diameter and reduce the transitivity. Transitivity is the probability that two neighbors of a node are connected. When the transitivity increases, the robustness does too. Li et al. [13] find that the botnet traffic comes from thousands of autonomous systems, i.e., different networks, which indicates that blacklisting of the botnet sources might not be sufficient to disrupt the botnet.

2.2 The Session Initiation Protocol

2.2.1 Introduction

The Session Initiation Protocol (SIP) is a text based protocol designed to negotiate the communication settings between multiple parties prior to establishing a communication session and transmitting the actual media. The communication settings are: network settings, transport protocol type, media type, bit rate and encoding. Once the communication has been signaled, then the media itself, i.e., voice, text, and video, is transmitted over another protocol; most often the Real-time Transport Protocol (RTP). The SIP role as a signaling protocol is to establish, modify, and terminate sessions. The SIP can work on top of different transport layer protocols, including User Datagram Protocol (UDP), Transmission Control Protocol (TCP), and Stream Control Transmission Protocol (SCTP).

SIP has been designed by the Internet Engineering Task Force (IETF) to be easy, scalable and flexible. It has been preferred by the Internet community over other signaling protocols, such as: H.323 developed by the International Telecommunication Union (ITU). Therefore, in November 2002, the 3rd Generation Partnership Project (3GPP) [8] adopted SIP as the standard signaling protocol in IP Multimedia Subsystem (IMS) to ease the integration and migration from circuit-based to IP-based network. 3GPP is the telecommunications standards body. SIP has been developed through several RFCs, but the most important two are: RFC3261 [14] which includes the core protocol specification, and RFC4083 [15] which includes the SIP extensions required by 3GPP.

2.2.2 SIP Message Structure

There are two types of SIP messages: request and response. A SIP request can be one of the following: REGISTER, to notify the SIP domain registrar about your location, INVITE, to establish a session, BYE, to terminate a session, CANCEL, to drop any unestablished session, OPTIONS, to ask another user/server about its capabilities, and MESSAGE, to send instant messages. SIP response is the ACK message which is used to confirm receiving a 200 OK reply. SIP full message consists of two parts separated by an empty line: header and body. All SIP messages, except INVITE and MESSAGE messages, do not need a body. A SIP user need an address/identifier in order to communicate with others, and his address

is written in a Uniform Resource Identifier (URI) format [16], such as sip:user@domain. The URI is similar the email address format.

The SIP message body is usually written in SDP: Session Description Protocol (SDP) format [17] to notify the callee about the type of the requested media, encodings, transport protocols. On the other hand, the body of the instant message request contains the text of the message.

The SIP message header consists of two parts; the first line and the rest of the header. The first line contains the request type, the Request-URI, and the SIP version. The Request-URI should have initially the same value as the To field, but it can be set to the next hop identifier as well. The first line in a response type message includes the response code. The response code can be a provisional (1xx), a success (2xx), a redirection (3xx), a client error (4xx), a server error (5xx), or a global failure (6xx). 2xx-6xx codes are called final responses. The rest of the message header consists of other fields related to the message type with corresponding values. Such fields include Via: specifies where should the recipient send any further response to the sender, From: caller identifier, To: callee identifier, Max-Forwards: similar to TTL in IP header, Call-ID: Call identifier, which is used with To and From fields to identify a dialog, or to identify all registrations of a particular user, CSeq: transaction identifier within a dialog, which helps maintaining the order, and identifies the request to which a particular response belongs, Contact: where the recipient should send any further request to the sender. To, From, CSeq, Call-ID, Max-Forwards, and Via header fields are mandatory in all SIP requests.

2.2.3 SIP Components

A SIP message is normally classified into 3 overlaying layers, each layer is a subset of the higher one. A Call consists of a one or more dialogs, and identified by the Call-ID header field value. A Dialog consists of a one or more transactions, and represents a relationship between two SIP users during a call. A Dialog starts with an INVITE request, and normally ends with a BYE request. In order to distinguish each dialog from the others within one call session, it is uniquely identified by three header field values: Call-ID, From, and To. The final and the lower layer is called the Transaction. The Transaction is identified by a CSeq header, and sometimes by a Branch parameter. The Branch parameter comes within the Via

header and identifies a branch of a transaction. The branch parameter is used by clients during a dialog establishment, and it is used by proxies to distinguish between different user agent responses to a forked request. The transaction consists of any request and all associated responses, such as the INVITE request and the associated responses until the ACK response. When a final response is 2xx then that ACK response is not considered part of the transaction. Another example of a transaction is the BYE request and the associated responses. Transaction states and times are maintained, and based on that a reliable SIP service can be provided over the User Datagram Protocol (UDP) protocol. Under this mechanism, a caller might choose to cancel or periodically retransmit the request until a response is received from the callee, then the caller send an ACK. The callee, on the other side, would periodically retransmit the final response until it receives the caller ACK.

A SIP network must have at least the following components: a User Agent (UA), a Proxy server, and a Registrar. Each domain has its own Proxy and Registrar servers. The SIP Proxy servers are used only to signal the communication. To improve network scalability, it is only the initial dialog signaling that actually goes through the SIP proxies, because later on, once both parties know the address of each other, then further dialog communications may bypass SIP proxies and both parties can communicate directly with each other. Each user knows about other's address via the Contact header field during the first INVITE handshakes. The MESSAGE request does not establish a dialog, therefore it needs always to go through the SIP proxies. The SIP message can be transmitted over User Datagram Protocol (UDP), Transmission Control Protocol (TCP), or Stream Control Transmission Protocol (SCTP) transport protocols.

The User Agent (UA) is the SIP client application that runs at the end systems, such as a user computer and a mobile phone. The UA is used to register, negotiate communication settings, establish a session, and terminate it. During a dialog, UA acts like a client and a server. The UA acts like a client when it sends a request or receives a response, aka User Agent Client (UAC). However, it acts like a server, when it receives a request or sends a response, aka User Agent Server (UAS). Callee user agent retransmits periodically the positive final response of an INVITE request, until it receives the Caller ACK.

For any two users to communicate, they have first to locate each other, and in order to do that, they have to register themselves with their domain registrar. This is done by sending a SIP REGISTER request to the domain Registrar via the proxy server. The Registrar extracts user information and current location from the request and stores them

in the domain's location DB service. Registration has to be refreshed periodically.

The Proxy server is responsible of receiving user requests and responses under its domain. It is also responsible for locating the receiver's proxy server by a DNS lookup, and routing the SIP message to it. The Receiver's proxy server locates the receiver by querying the domain location DB service, and then forwards the message to him. The Proxy server can be stateful or stateless. The Stateful proxy is the default configuration despite the performance limitation to maintain accounting, forking, and other services.

2.2.4 SIP Extensions

On November 2002; 3rd Generation Partnership Project (3GPP) [8] has adopted SIP to be the multimedia signaling protocol for IP Multimedia Subsystem (IMS), i.e., mobile telecommunication core networks. Therefore, 3GPP has extended many aspects of SIP specifications, added new features required by IMS, and fixed some security flows. SIP requirements identified by 3GPP Release 5 IMS are documented by IETF in RFC4083 [15], and they are under analysis.

SIP is originally designed to be end-2-end clear text signaling protocol, where SIP network elements are only needed to locate both parties, which happens to be at the beginning of each dialog, other than that, both ends communicate directly with each other, which avoids single point of failure and improves SIP network performance & scalability.

On IMS, a user may have one private identity and multiple public identities. User agent is authenticated using a shared secret known only to the user device and the service provider. SIP communication is encrypted by Internet Protocol Security(IPSec)/Transport Layer Security(TLS). User communications have to go through IMS components, i.e., via Call Session Control Function (CSCF) components. CSCF components protect the security and the privacy of the networks and the users. The privacy is protected by enforcing the communication to follow specific paths, and controlling the traversed information by masking some header fields and adding others. All user communications have to go through IMS access point Proxy-CSCF (P-CSCF), which is the closest component to the Proxy server in SIP regular network, and IMS registrar is called Serving-CSCF (S-CSCF). The P-CSCF prevents unauthorized users from accessing the network, by associating secure ports to each registered user agent. In addition, P-CSCF enforces a strict route according to the reserved

route during the registration process, which prevents session hijacking.

2.3 Exploiting SIP for Creating Botnets

2.3.1 Why is SIP attractive as a Command and Control protocol?

According to a study done by Berger et al. [18], SIP has outstanding features that nominate it as useful Command and Control protocol. Unlike the P2P protocols, SIP, as a standard voice over IP signaling protocol, is expected to have less restricted access policy, similar to HTTP, DNS, and SMTP access policy. The SIP infrastructure provides botmaster with some management tools, such as the registration mechanism, keeping track of client status afterward, and providing reliable message delivery mechanism. The registration process of the SIP identifier, i.e., the Uniform Resource Identifier (URI), helps maintain a dynamic list of user name translations. Therefore, it helps maintain the reachability to SIP users regardless of their real IP addresses. The SIP Presence service [19] is used to keep track of user availabilities, where a bot can subscribe to a SIP presence Agent/Server to be notified about the availabilities of other users. Such a service is valuable, because otherwise very noisy and frequent short live status notification/request messages will be exchanged. Finally, SIP message structure provides many options to conduct command and control communications, such as SIP instant messaging, message standard header, message user-defined header, and message body.

2.3.2 How it works?

In order to hide the Command and Control data or pass them to the bots, a botmaster can choose from multiple options or use them all randomly to obfuscate detection effort [18]. One direct option to use is to establish a SIP session first, then exchange information and instructions in between. The overhead and the noise is high in such option. In addition, the accessibility to the ports that will be used by the further established communication (such as RTP) is not guaranteed since SIP is only a signaling protocol. However, this option mimics the pattern of the normal SIP traffic, especially if the length of sessions are randomized. Another simpler, lighter, and lower overhead option is to use SIP Instant Messaging functionality and Reliable delivery [20] to exchange Command and Control information. The third option is to use the Request/Response messages themselves to pass Command

and Control information. Keep in mind that abnormal patterns can expose the botnet. An example of an abnormal pattern is when a caller sends INVITE requests, and the callee rejects them all.

A SIP message has many options that can be utilized to carry Command and Control information, such as: standard message header, message body, or user-defined header. The last option is the weakest option among the three as it would carry clear keys that can be used for detection.

2.4 Related Work

There are different approaches for botnet detection and mitigation. They can be summarized into the following categories [13]. The first approach is analyzing a bot source code to understand how it works and how to stop it. The source code is not always available [4,13]. Another approach is to understand how a botnet works by installing a honeypot/honeynet. A honeypot is a fake bot that attracts botnet infections in order to observe the botnet activities. The third approach is the signature based detection approach. Botnet signatures can be discovered by any of the previous two approaches, i.e., the source code and the honeypot approaches. All the three previous categories have to be carried on frequently whenever a new bot or a new version of an existing one exists. The fourth and final category is anomaly based detection, which can be classified further into two subcategories: the first is to detect a botnet based on an abnormal behavior such as a high volume traffic, which can indicate potential botnet malicious activities, and the second subcategory is to detect a botnet based on Command and Control communications, which is the recent and recommended approach [6, 7, 12, 21]. We provide more details on this approach in the following subsection.

2.4.1 Command and Control Detection Approaches

The Command and Control channel is the anchor point, and the middle joint between all botnet components, because it is the control medium. Therefore, it is the best approach to mitigate botnet threats. By monitoring and detecting the Command and Control channel it is possible to intercept botnet communications at early stages, hopefully before launching any attack. By detecting the Command and Control channel, a defender can detect both the controllers and the bots that communicate with them. By blocking access to botnet

controllers we disconnect the botmasters from their bots. Therefore, we disrupt botnet managements [6, 7, 12, 21]. Detecting Command and Control channels is challenging, but achievable based on a key principle that bots within a botnet tend to behave in a similar or correlated manner [6, 22].

Strayer et al. [5] conducted the first work and proposed an IRC botnet detection model based on the network behavior in the Command and Control communication channel. The authors identify the Command and Control activities by characterizing the flow based on number of parameters, such as bandwidth, packet timing, and burst duration. The recorded traces are filtered out through number of steps. Then, the system classifies and correlates between the remaining flows. The distance between 2 flows are calculated by this formula $D = \sqrt{\sum_{i=1}^{n}(norm\_diff_i)^2}$, where n is the number of attributes, $norm\_diff_i$ is the normalized difference between the i^{th} attribute in 2 flows ($\frac{|v_1 - v_2|}{v_1 + v_2}$). The goal was to find the flow pairs with the very small distance (closes to zero), because it'd indicate they were part of the same Command and Control channel. The flaws that successfully pass the classification and the correlation phase are almost but not guaranteed part of botnet activities. Finally, the system does a topological analysis to find out the controller. This is done by drawing a directed graph based on the similar flows, then selecting the highest in-degree/out-degree node.

Gu et al. [6] recognized that bots within a botnet usually share common preprogrammed steps. The preprogrammed steps create a traceable patterns, therefore, they noticed that bots behave in a crowd and similar manner. The authors propose BotSniffer system. The proposed system detects a centralized Command and Control channel over two protocols: IRC which is a Push style, and HTTP, that is a Pull style. It consists of a snort and network-based analysis components. The proposed system identifies those bots that do similar Command and Control and malicious activities within limited variance on sequence, and time shift. BotSniffer looks into the density and homogeneity of response crowd that comes from clients connecting to the same HTTP/IRC server, which is computed using DICE coefficient based on the response message content. When the accumulated likelihood of correlation between set of bots exceeds a given threshold, they are declared bots.

To overcome the limitations of previous work such as the botnet structure/protocol dependent and some detection difficulties of crowd homogeneity based on encrypted message content, Gu et al. [22] propose BotMiner which is a structure/protocol independent network detection system. The BotMiner captures all TCP/UDP flows and aggregates them into

fixed number of flows during a specific time window. The flows are aggregated by applying quartiles(binning) technique. After aggregation, the system applies two X-means steps to classify users. X-means is a clustering technique. Those users who share both similar Command and Control communication patterns and similar malicious activity patterns are clustered together and considered members of the same botnet.

Yu et al. [7] propose an online and a content-independent detection framework to detect botnet activities. Each traffic flow is represented by a set of characteristics, such as: Start/End time, duration, IP protocol, Number of packets, and Total number of bytes. The similarity between flow characteristics are measured using incremental Discrete Fourier Transform (DFT) technique. The flow pairs with high similarities are added to a list of suspected hosts. The suspected hosts are monitored for a specific amount of time for botnet malicious activities, such as a scanning and a spamming using Snort [23] plugins. When ever a suspected host commit at least one botnet malicious activity, it is then classified as a bot, otherwise, a host is removed from the suspected list if the observation time is over. Similarly but simpler, Zeidanloo et al. [21] propose a general and content-independent P2P Botnet detection framework, and the system consists of a couple of components. There is a Traffic Monitoring component to detect hosts behaving similarly, by comparing the flows characteristics. Another component is a Malicious activity detector which does exactly as the name state. The last component is an Analyzer which combines the results from the other two components, and the hosts that exist in both components are classified as bots.

2.4.2 SIP Botnet Detection

Berger et al. [18] designed and implemented a SIP botnet prototype, aka SipBot, based on Storm botnet communication structure, where Overnet protocol was replaced by a SIP protocol. The authors suggest the following points to improve the prototype and to have a quieter SIP botnet implementation: switching to push mode, using SIP presence capabilities for availability status, and aligning its behavior to be within the boundaries of natural SIP communication patterns. We are not aware of previous works on detecting SIP botnets.

2.5 Summary

The Command and Control channel is the weakest point in botnets. Thus by blocking access to it, the botmaster is prevented from controlling botnet members. Due to the dynamic

nature of botnets, the most successful detection approach is the one that neither relies on specific Command and Control channel settings, nor on keywords. This is because both can be easily modified. Therefore, monitoring the user behavior is the best approach to detect botnets Command and Control channel.

Chapter 3

Proposed SIP Botnet Detection System

3.1 Overview

In this chapter, we propose an online botnet detection system, designed based on the following factors: bots tend to behave in a similar or a correlated manner [6,22] and the Command and Control channel is the botnet weakest point [6, 10]. The proposed system can detect botnets at early stages even before they initiate any malicious activities. The proposed system does not rely on the structure of the Command and Control channel. Thus, it works for centralized and distributed botnets. In addition, the proposed system does not use the contents of the SIP messages, which makes it not limited to specific botnet implementations. Furthermore, the proposed system does not rely on the sequence of events, which enables it to resist time evasion techniques.

In order to monitor any traffic coming in or going out to/from a network X, The proposed system should be installed between X's gateway and the internal network, as shown in Figure 3.1.

The proposed system consists of two main components, Monitoring engine and Correlation & Detection engine, as shown in Figure 3.2. The main role of the monitoring engine is to capture and log the designated traffic. The role of the Correlation & Detection engine is to detect bots, and identify potential botnet controllers.

The proposed system functions as follows:

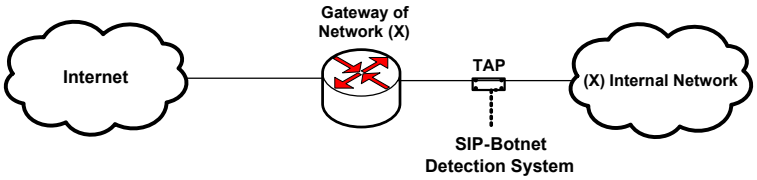


Figure 3.1: The proposed system is installed on this location to monitor network X.

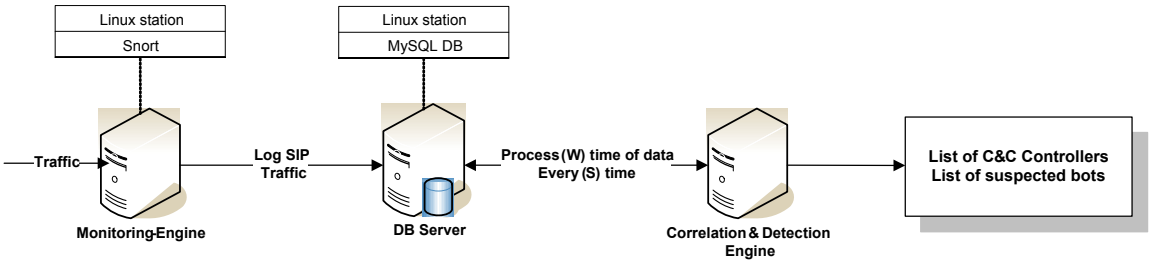


Figure 3.2: Components of the proposed SIP botnet detection system.

- SIP communications, i.e., Command and Control communications are captured by the monitoring engine.
- Logs collected from SIP communications are transferred into feature vectors.
- A similarity algorithm classifies users based on their feature vectors. This classification results in one or more groups of similar users.
- Similar users are identified as suspected bots. Then the SIP botnet Controllers are identified based on a directed graph representation of the suspected bot communications.
- Both lists of suspected bots and controllers are reported to the administrator.

The following sections provide more details on the above steps.

3.2 The Monitoring Engine

The monitoring Engine captures SIP traffic and logs it to a central database server. The monitoring engine consists of monitoring and logging agents, and a MySQL database server.

```
# $Id: local.rules,v 1.11 2004/07/23 20:15:44 bmc Exp $
# ----------------
# LOCAL RULES
# ----------------

# Definitions:
portvar SIP 5060

# Catch C&C:
#
# 1. Catch any SIP communications from the monitored network.
log udp $HOME_NET any -> any $SIP (sid:100000;)

# 2. Catch any SIP communications to the monitored network.
log udp !$HOME_NET any -> $HOME_NET $SIP (sid:200000;)
```

Figure 3.3: Snort rule configuration to capture SIP traffic.

In order to eliminate any potential overhead on network routers, we propose to setup the monitoring engine on a standalone system. As shown on Figure 3.1, the traffic can be directed to our monitoring engine by installing a network TAP without imposing any risk or overhead on the network. A TAP is a hardware layer device that provides access to a traffic without imposing any overhead on the network and often used by intrusion detection systems. For the purpose of the monitoring and logging functions, we use Snort, which is an open source intrusion detection system [23]. Capturing the whole packet by snort is necessary for our work, as tracking SIP sessions relies on some SIP headers information. Snort can be installed on a standalone machine and supports logging to MySQL server. We applied a set of Snort rules to capture any SIP communications coming in or out the monitored network. These rules are listed in Figure 3.3.

Snort listens on the network interface and logs any captured traffic to a central MySQL database server. The MySQL database is installed on another standalone machine to serve as a central database server, in case there are more than one logging agents. A customized database is created to receive the recorded traffic, following instructions suggested by Snort team.

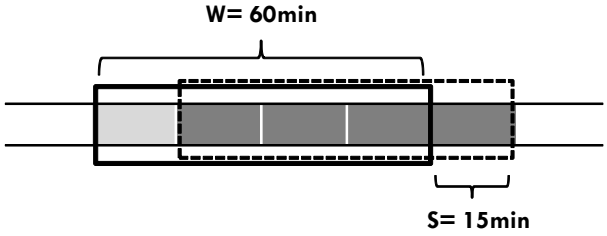


Figure 3.4: The time window W slides forwards for S min.

3.3 The Correlation and Detection Engine

3.3.1 Representing user SIP sessions

Constructing SIP sessions requires at least the value of the Call-id field [14]. Tracking the source/destination IP and port is not useful, because SIP uses port 5060 by default for both source and destination.

In order to measure the similarity between users, we should be able to measure the similarity between users' activities. Therefore, it is important to represent the semantic of user activities by series of unique features. To do that, we need first to identify such features, and extract them from the raw session representations. The list of unique features that we choose are:

- Duration of the session in seconds.
- Total size of the session in bytes.
- Number of packets in the session.
- Average Bytes per second (Bps) in the session.
- Average Bytes per packet (Bpp) in the session.

We transfer the extracted features into a Feature Vector (FV). We collect the feature vectors of all SIP sessions conducted by each user during a time window W. A set of user sessions during W are arranged in a Feature Stream (FS). The feature stream is a matrix representation holding all the feature vectors, as shown in Figure 3.5. The time window W consists of smaller and equal size of sliding windows S. The time window W moves forward periodically by S minutes, as shown in Figure 3.4. At the end, the users that have sessions during W are represented by their feature streams.

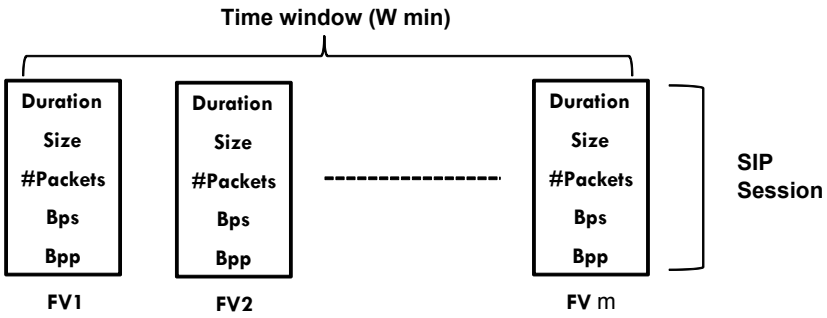


Figure 3.5: SIP sessions during a time window W are represented by a Feature Stream (FS). Each SIP session is represented by a Feature Vector (FV).

The functionality details of the Correlation and Detection engine are on the following subsections, but can be summarized as follows:

- Detecting similar users who are active during the past time window W, by measuring the similarity between them.
- The lists of similar users are identified as suspected bots since bots tend to behave in similar or correlated manner [6, 22].
- The list of potential controllers is generated based on that.
- Both lists are reported to the network administrator.

The network administrator put the list of the suspected bots in the watch list of his network detection system. As soon as a suspected bot initiates what is classified as botnet malicious activity such as spamming, phishing, and denial of service attack [2, 3], then the network administrator can confirm the identity of that suspected bot. As an extension to the work of this project, monitoring the malicious activities can be integrated into the proposed system.

3.3.2 Measuring Similarity among users

The similarity between two sessions are measured by the normalized euclidean distance between their feature vector representations FV_1 and FV_2. If FV_1 represents a feature vector for $user_1$, and FV_2 represents a feature vector for $user_2$, then the normalized euclidean distance between the two feature vectors is calculated as:

$$D(FV_2, FV_1) = \sqrt{\sum_{i=1}^{f} (\frac{|FV_2[i] - FV_1[i]|}{FV_2[i] + FV_1[i]})^2} \tag{3.1}$$

where f is the number of features in a feature vector. The euclidean distance has been normalized to bound the distance to a maximum value of f [5]. Two sessions are considered similar if the distance between them is $\leq \alpha$. α is a threshold used to measure the distance between two sessions.

The similarity between two users U_1 and U_2 is measured by calculating the Jaccard index between their feature stream representations FS_1 and FS_2, also known as Jaccard similarity coefficient, as follows:

$$J(U_1, U_2) = \frac{|FS_1 \cap FS_2|}{|FS_1 \cup FS_2|} \tag{3.2}$$

$$|FS_1 \cup FS_2| = |FS_1 \cap FS_2| + |FS_1 \Delta FS_2| \tag{3.3}$$

To measure the size of the intersection set, we count the similar feature vectors between the two users. To find out the similar feature vectors, we measure the distance between them. Therefore, if each user has a maximum of M feature vectors, then the normalized euclidean distance is calculated M^2 times between all feature vectors in both users. This means calculating the distance between each feature vector in one user with all feature vectors in the other one. The size of the Union set is calculated as shown in Eq.(3.3), which includes the size of the intersection set and the total number of dissimilar feature vectors in both users.

As shown in Figure 3.6, the more two users have in common, the more similar they are. Two users are considered similar if the Jaccard index value is $\geq \beta$. β is a threshold used to measure the similarity between two users. The process of measuring the similarity between all users is a combination without repetition, also called Binomial Coefficient, where N=number of users, r=2 users, and the number of the required similarity checks is $C_2^N = \frac{N \times (N-1)}{2}$, So generally speaking, the total time complexity of the proposed algorithm is:

$$C_2^N = \frac{N \times (N-1)}{2} \times M^2 = O(N^2 M^2) \tag{3.4}$$

where N is the number of users, and M is the number of user sessions.

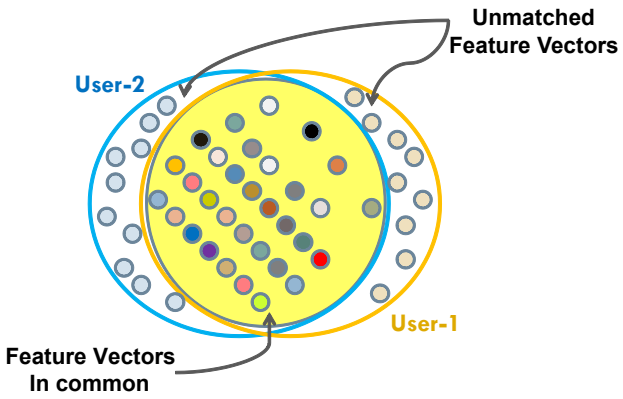


Figure 3.6: Graphic Representation of the similarity between users. For two users User-1 and User-2, the more feature vectors they have in common, the more similar they are to each other.

In addition to that, The proposed algorithm is implemented in two modes; the normal mode which what we have explained before, and means that, at every sliding window S, the system constructs and compares the user sessions of the past time window W minutes. The time complexity of the normal mode is shown in Eq.(3.4). Figure 3.7 shows a pseudocode of the normal mode implementation. The second is the incremental mode, where the system mainly compares only the recent users and sessions from the new S window with the available users and sessions from the past W. Since S is typically much smaller than W, and the amount of new data is only $\frac{S}{W}$ of the full amount of data during W, then the time complexity of the incremental mode is much smaller too. The time complexity of the incremental mode is $O((\frac{S}{W})^2 N^2 M^2)$, where $(\frac{S}{W})$ represents the proportional amount of the new data during S to the total amount of data during the whole W. Figure 3.8 shows a pseudocode of the incremental mode implementation.

We conducted some experiments to evaluate the time required to measure the similarity between users. The computations took place on a laptop with 2.4GHz CPU and 2GB RAM. The results in Figure 3.9 show that the computations can easily be performed in both modes by any reasonable servers. The results show too that the incremental mode perform much better than the normal mode. Figure 3.9 shows under the normal cost, that the average execution time is about 1min, when the number of active sessions is about 6500, i.e., 9sec for each 1000 sessions. The figure shows also that the execution time is about 12.8min, when the number of active sessions is 32000, i.e., $\approx$ 23.5sec for each 1000 sessions, which means a 2.3 times the time required to process 1000 sessions in the first case. On the other

```
While running do
    Initialize all data structures
    Read IP flows for the past W min
    Create/Update SIP sessions
    for user i = 1 to n do
        for user j=i+1 to n do
            if Similarity(user[i], user[j]) > beta then
                add them to the bots list
            end if
        end for
    end for
    Identify botnet controllers
    Report bot and controller lists to the Administrator
    Sleep for S min
End while
```

Figure 3.7: Pseudocode of Normal mode implementation.

```
While running do
    If the age of the buffered data > W min then
        Slide data buffers forward by S min, to get rid of old data
    end if

    Read IP flows for the past S min
    Create/Update SIP sessions

    for user i = 1 to n do
        for user j=i+1 to n do
            if at least one user is new then
                if Sim(user[i], user[j]) > beta then
                    add it to the bots list
                end if
            else if Sim_incremental (user[i], user[j]) > beta then
                add it to the bots list
            end else if
        end for
    end for
    Identify botnet controllers
    Report bot and controller lists to the Administrator
    Sleep for S min
End while
```

Figure 3.8: Pseudocode of Incremental mode implementation.

hand, Figure 3.9 shows under the incremental cost that the average execution time required to process the whole 32000 sessions together is only 50sec. The same figure shows that the incremental mode manages to reduce the time by 68% to 93%.

3.3.3 Detecting SIP Botnet Controllers

Once we detect the lists of suspected bots, the next step is to identify botnet controllers which helps disconnecting the Command and Control channel between the botmaster and the bots. In order to find out the botnet controllers, we draw a directed graph based on the sessions that carried on by the recent list of bots, where a directed edge is drawn between a bot and a destination node if the bot initiates at least one connection to that node. Once the directed graph is drawn, then the list of controllers are constructed from those nodes that have in-degree $> \delta$.

Setting the value of δ is a bit tricky to avoid lot of false positives and false negatives. Knowing that controllers should receive a noticeable amount of connections from different bots much higher than any other bots, then we preliminary set δ as:

$$\delta = (\frac{\text{Number-of-bots}}{10}) \tag{3.5}$$

i.e., we define the controller as a bot that has been contacted by more than 10% of the bots.

Alternatively, we can sort the list of bots according to their in-degree values, then pick number of nodes that have the highest in-degree, and identify them as controllers. However, we would still need a minimum bound to avoid false positives.

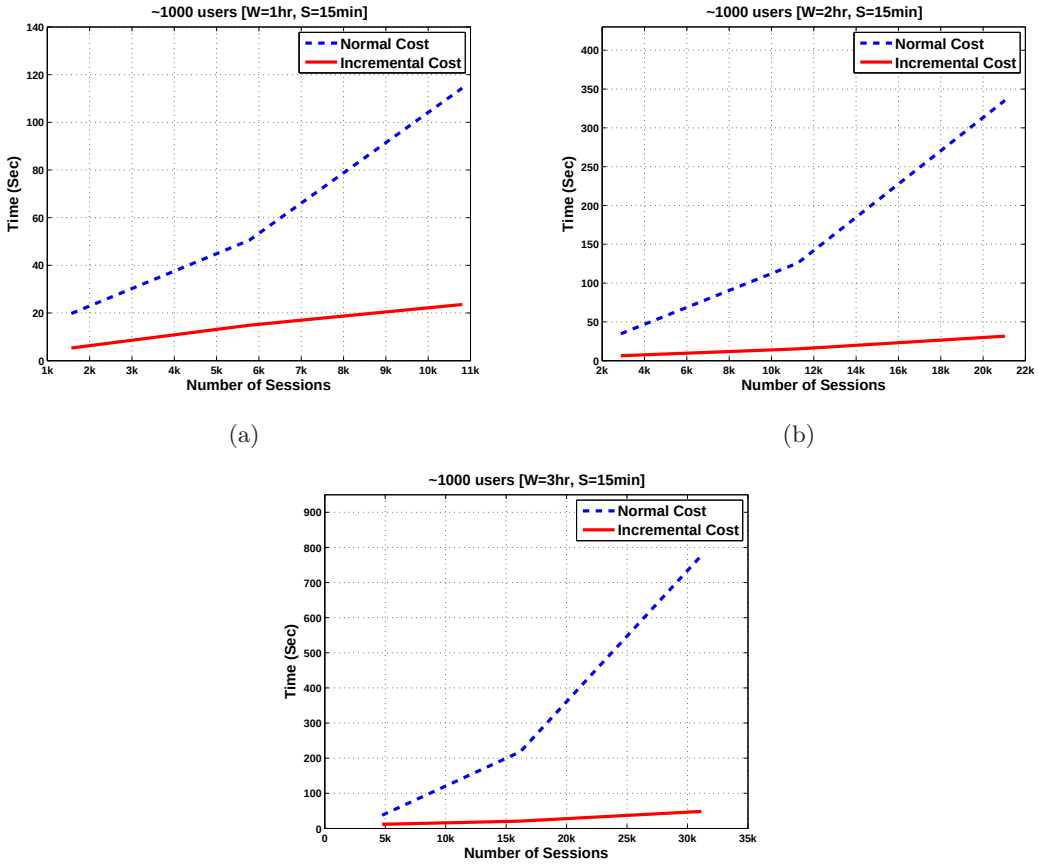


Figure 3.9: Average Execution Times at Normal & Incremental Modes when W=1hr/2hr/3hr. Each figure shows the Average Execution Times at different number of sessions. Number of sessions increases as number of bots increases, i.e., 10/50/100 bots. Incremental mode shows a much less cost than the normal mode implementation.

Chapter 4

Experimental Evaluation

In order to evaluate precision of the proposed system, we built a test network at the Network Systems Lab, SFU-Surrey. We evaluated the precision based on an emulated traffic generated by two tools: autosip and sipbot [18], and several shell scripts that help configuring virtual interfaces, generating some configuration/contact files, and automating the process. We measured the precision by computing the False Positive rate (FP) and False Negative rate (FN). The experiments show that the proposed system achieves high precision with low FP $\approx 4.7\%$, and almost zero FN.

4.1 Datasets

We collected data sets for our experiments from an emulated traffic generated by two C/C++ tools: autosip and sipbot [18].

4.1.1 Autosip software

Autosip emulates a realistic behavior of regular user calls by modeling various characteristics, such as the number of on-line users should vary within time, call duration follows a log-normal distribution [24], a user calls a friend with probability ρ and others with probability $(1-\rho)$, and the final variable is that a user makes on average χ calls uniformly distributed over the hour, and a call probability per minute is $\frac{\chi}{60}$. The tool has two components: a manager and a client. The manager component sets call parameters, distributes them among users, and controls the number of on-line users during the time. The manager

component comes with an interface used for configuring call parameters, providing status, and starting/stopping the emulation. At the beginning, Autosip clients connect to the manager and get the call parameters, then wait until they are executed, i.e., activated by the manager. Active clients register, then sleep for a random time. Afterward, they start calling other users using the contacts file.

4.1.2 Sipbot software

In order to generate SIP Botnet traffic, Berger et al. [18] developed Sipbot tool on top of the Stormnet, which is a P2P botnet [25]. The Overnet protocol used in the Stormnet has been replaced by the SIP. As a proof of concept, the authors emulate SIP Command and Control communications by sending SIP INVITE message from the bots to the controllers, and controllers in return respond with 603 Decline without establishing a session. Sipbot does not have a manager component, and clients simply register then call each other.

4.2 Testbed Setup

The testbed is consisted of the following components, as shown in Figure 4.1:

- A MySQL Data Base (DB): A central log server installed on Ubuntu Linux station.
- Virtual bots: Sipbot tool has been used to emulate bot activities over SIP. It is installed on another Ubuntu Linux station. Each bot is given a virtual IP. As part of the monitoring engine, Snort [23] has been installed on the same station to capture the designated SIP traffic, and log it into the central MySQL database. Opensips, which is an open source implementation of a SIP server [26], has been installed on the same station as well to act as a SIP Registrar for SIP bots.
- Virtual users: Autosip tool has been used to emulate normal user activities over SIP. It is installed on two Ubuntu Linux stations. Each user is given a virtual IP. Same as previous component, Snort [23] has been installed on both stations to capture the designated SIP traffic, and log it into the central MySQL database. Opensips as well has been installed to work as a SIP Registrar for SIP users.

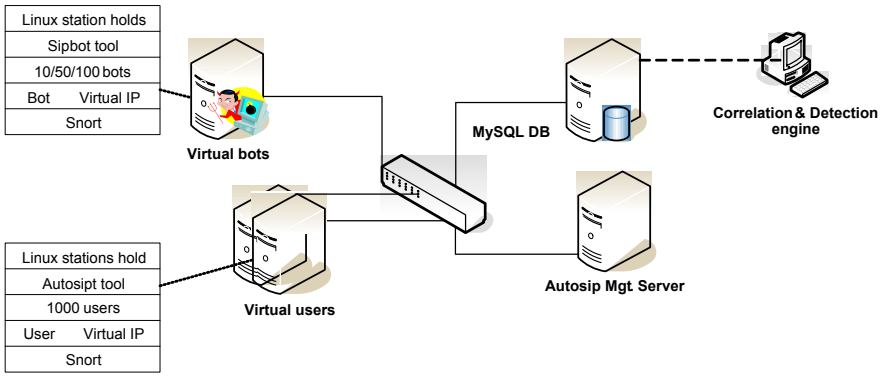


Figure 4.1: Testbed Setup

- Autosip management server: A server used to manage the virtual users. For simpler network context, the autosip manager is installed on a separate Ubuntu station, although it is possible to have it installed on one of the Autosip user stations.
- Correlation & Detection engine: It detect bots and identify controllers. It is installed on external laptop that has access to the MySQL database server.

4.3 Performance Metrics and Experimental Setup

We measured the precision of the algorithm by two metrics:

- False Positive rate (FP), which is the ratio of normal users that have been identified as suspected bots to the total number of active users.
- False Negative rate (FN), which is the ratio of bots that have not been detected to the total number of active bots.

We conducted five types of experiments as follows:

- Set-A: Tuning thresholds α and β by fixing the time window W at 3hr and sliding window S at 15min, while changing α and β.
- Four types of large scale experiments to generate different scales of data and to verify the precision of the proposed system. The experiments lasted for 24 hours with 1000

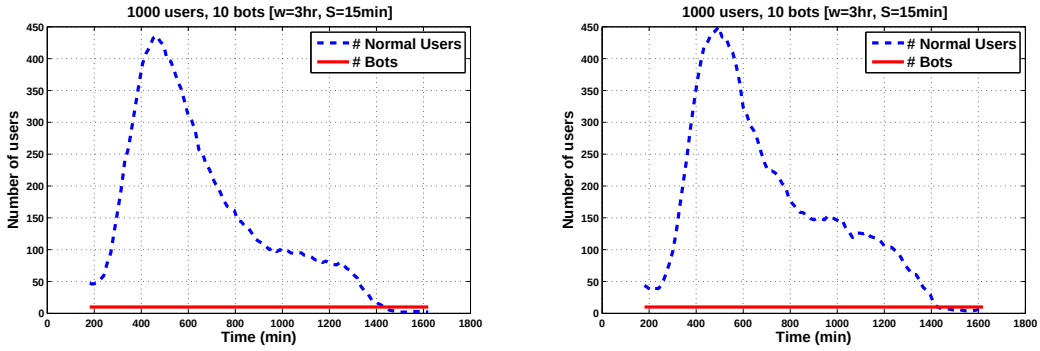
Index	Type	Description
Set-A	Thresholds Tuning	Tune α and β.
Set-B	Compute FP/FN	At 1000 users, 10 bots, and W= 2hr and 3hr.
Set-C	Compute FP/FN	At 1000 users, 50 bots, and W= 2hr and 3hr.
Set-D	Compute FP/FN	At 1000 users, 100 bots, and W= 2hr and 3hr.
Set-E	Compute FP/FN	At W= 1hr, 1000 users, and 10, 50, and 100 bots.

Table 4.1: Summary of the conducted experiments.

users and different number of bots proportional to the number of normal users: 10 bots (1%), 50 bots (5%), and 100 bots (10%). We conducted the exepriments at different sizes of time window W and sliding window S, in order to find out the optimum window sizes required to achieve the best precision. We selected only 3 time window sizes W= 1hr, 2hr, and 3hr. We think that the 3hr time window size represents a fair upper bound. It is long enough to capture reasonable amount of botnet activities. We think that any larger size would be unjustifiable, specially with the memory overhead and the extra execution time that it would impose. The four types of experminets are as follows, as shown in Table 4.1:

- Set-B: At 1000 users and 10 bots; we conducted two types of experiments at window size W= 2 and 3 hours. For each time window, we conducted 6 experiments at the following sliding window S: 5min, 10min, 15min, 20min, 25min, and 30min.
- Set-C: We conducted a similar set of experiments of Set-B at 1000 users and 50 bots, but with only two sliding window S at 15min and 30min, due to the longer execution time, as shown in Figure 3.9. Both S values represent the medium and the maximum sliding window sizes, as 5min is too short and would increase the overhead dramatically, and the distance between 10min and 15min is short too.
- Set-D: We conducted a similar set of experiments of Set-C at 1000 users and 100 bots.
- Set-E: At window size W= 1 hour, and sliding window S= 15min, we conducted three types of experiments at 1000 users and 10, 50, and 100 bots.

Prior to previous types of experiments, a preliminary step has been done to verify the correctness of the input data sets. The following subsections have more details.

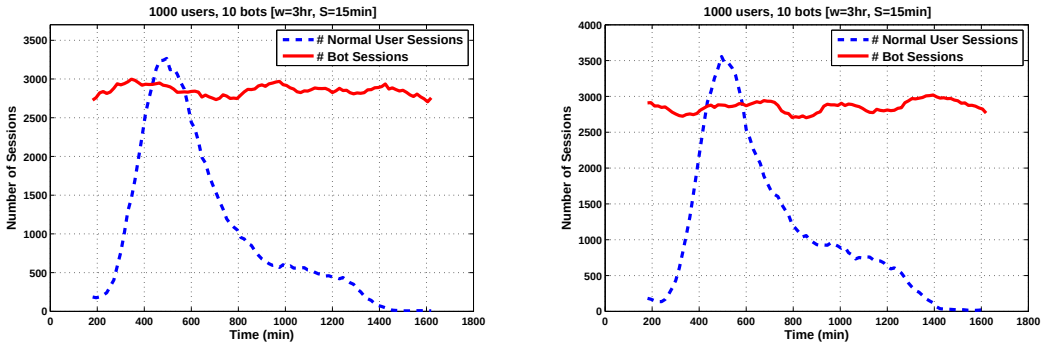


(a) (b)

Figure 4.2: Number of users from two experiments. Both look alike and shows a dynamic increase & decrease on the number of users during time.

4.3.1 Verifying Correctness of Input Data Sets

In order to be able to carry on the experimental evaluations as planned, we adjusted the system to log all the required information which includes the following: α, β, window-start-time, window-end-time, elapsed time since the beginning of the log, execution time, total number of active users, number of active normal users, number of active bots, total number of active sessions, number of active users sessions, number of active bots sessions, number of false negatives, FN rate, number of false positives, and FP rate. We generated two data sets from 1000 users and 10 bots communicating for 24 hours on two different days. The time window W was set to 3 hours and S to 15 minutes. We shed some light previously on the two tools that generated the input data sets. More details about their reliability is found in [24]. What we'd like to show here is that the input data sets show a dynamic and a consistent nature. The number of active users is shown in Figure 4.2. Both figures look alike and the number of active users varies dynamically through time. The number of sessions is shown in Figure 4.3. The number of active sessions increases or decreases according to the number of active users during time.



(a) (b)

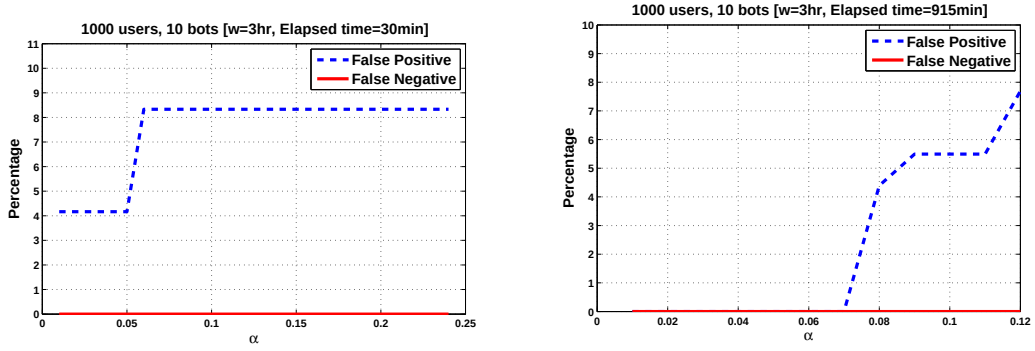
Figure 4.3: Number of sessions from two experiments with the same settings. Both look alike and show increase & decrease on the number of sessions during time.

.

4.3.2 Set-A experiments: Tuning Thresholds

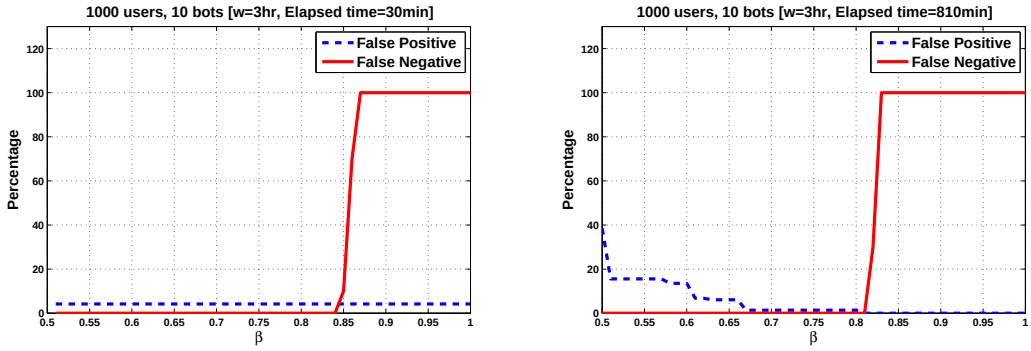
There are two threshold variables; α which helps determining if two sessions for two users are alike, and β which helps determining if two users are alike. To be able to compute FP/FN, it is necessary to have the optimum threshold values. Before tuning threshold values, it is necessary to fix the other variables, like: time window size W at 3hr and Sliding Window size S at 15min. The reason to why we sat W to 3hr is because some preliminary experiments showed to us that the proposed system has achieved better precision when the W= 3hr, so we anticipated that the best setting to tune our thresholds with is when W= 3hr. Finally we picked specific elapsed times at 30 min, 915 min, and 810 min, where FP/FN had together the worst values.

Based on the previous two generated traffic (1000 users, 10 bots, W=3hr, and S=15m), on specific elapsed times (30 min, 915 min, and 810 min), we first tuned α because it does not rely on any other factors. We found that the optimum value, within [0.01 - 1.0] range, is when α= 0.05, as shown in Figure 4.4, i.e. the difference between two similar sessions must not exceed that. Next, we tuned β given that the optimum value of α is 0.05, and found that the optimum value, within [0.51 - 1.0] range, is when β= 0.8, as shown in Figure 4.5, which means two users are considered similar if they have in common at least 80% of their sessions, which is a very reasonable value that is not too high which might cause false negatives, or too low that might cause higher false positives.



(a) α (b) α

Figure 4.4: Results for Set-A experiments (α). α has been tuned on two different times where FP/FN were together the worst. The optimum value is 0.05 based on (a). Obviously, false negative does not increase as α increases.



(a) β (b) β

Figure 4.5: Results for Set-A experiments (β). β has been tuned on two different times where FP/FN were together the worst. The optimum value is 0.8 based on both graphs.

4.4 Experiment Results for Algorithm Precision (FP/FN)

Given α=0.05 and β=0.8; the following sub sections presents the results of the four experiment types that described before.

4.4.1 Set-B experiment Results

At 1000 users and 10 bots, FP/FN have been calculated at W=3hr and 2hr, for 6 different sizes of sliding window S at 5min, 10min, 15min, 20min, 25min, and 30min. The results show that regardless of the sliding window size, the maximum FP is 10.8%, when Win=2hr, as shown in Figure 4.6, and it is 4.7%, when Win=3hr, as shown in Figure 4.7. On the other hand, FN is zero when W=3hr, and demonstrates some few spikes when W= 2hr, but only in 2-3 consecutive processing times across different sliding window sizes, which indicates that they are related to the traffic nature at that time. Therefore the optimum precision values are achieved at the time window W= 3hr.

4.4.2 Set-C experiment Results

At 1000 users and 50 bots, FP/FN have been calculated at W=3hr and 2hr. Only two sliding windows S have been applied, due to a longer execution time. The two S are 15min and 30min, which are the medium and the maximum sizes. The results show that FN is always zero. It shows that the maximum FP is 6.3% when W= 2hr, and it is 2.7% when W=3hr, as shown in Figure 4.8.

4.4.3 Set-D experiment Results

At 1000 users and 100 bots, FP/FN has been calculated at W= 3hr and 2hr. Only two sliding windows S have been applied at 15min and 30min. The results show that the maximum FP is 1.7% when W= 2hr, and it is 2.1% when W= 3hr. FN is always zero, as shown in Figure 4.9.

4.4.4 Set-E experiment Results

To demonstrate inefficiency of setting W to 1hr, we conducted three types of experiments at window size W= 1hr, and sliding window S= 15min for 1000 users and 10, 50, and 100 bots. Figure 4.10 shows that the maximum FP is 22% at Bots=10, and the next maximum

is 18% at Bots=50, which means 7%-11% higher than the maximum FP on the previous two window sizes, i.e., 2hr and 3hr. FN is 0% at Bots = 50 and 100, and unstable at Bots = 10.

4.4.5 Results Summary of the proposed system

From all above experiments, we conclude that the proposed system performs the best when α= 0.05, and β= 0.8. Among the three time window sizes, the proposed system achieves the lowest FP and FN across all experiments when the window size W= 3hr, and the worst FP and FN when the windows size W= 1hr. This is because when W= 1hr there is no enough data to reliably estimate similarity among users. On the otherhand, various sizes of the sliding window S that have been applied do not show any clear impact on the FP and FN. Therefore, all of them are applicable. We notice that the FP seemed to decrease when the number of bots increased, so we looked closer to the actual log data to have a better understanding. Before trying to justify the previous behavior, we'd like first to clarify that the number of active users at any time , the number of their activities, and the nature of them are randomly and dynamically changing. They are done on this way to emulate the normal user behaviors. Therefore, they are varied from one experiment to another. Moreover, each behavior has different type of explanation. From a general overview of the data, we notice that the actual number of false positives are generally close to each other across all different number of bots experiments. What makes the false positive some times smaller or larger is the number of active users at that time. This is the only reason that we can think of and let the 100 bot experiments have smaller false positives. By looking to the 50 bot experiments, on the other hand, we notice that the number of active users and their activities are generally much larger than the number of active users and their activities in the 10 bot experiments. The 50 bot experiments took place during the day, and the 10 bot experiments took place during the night. Having users who are more active, provides the system with more data that leads to a better similarity measurement.

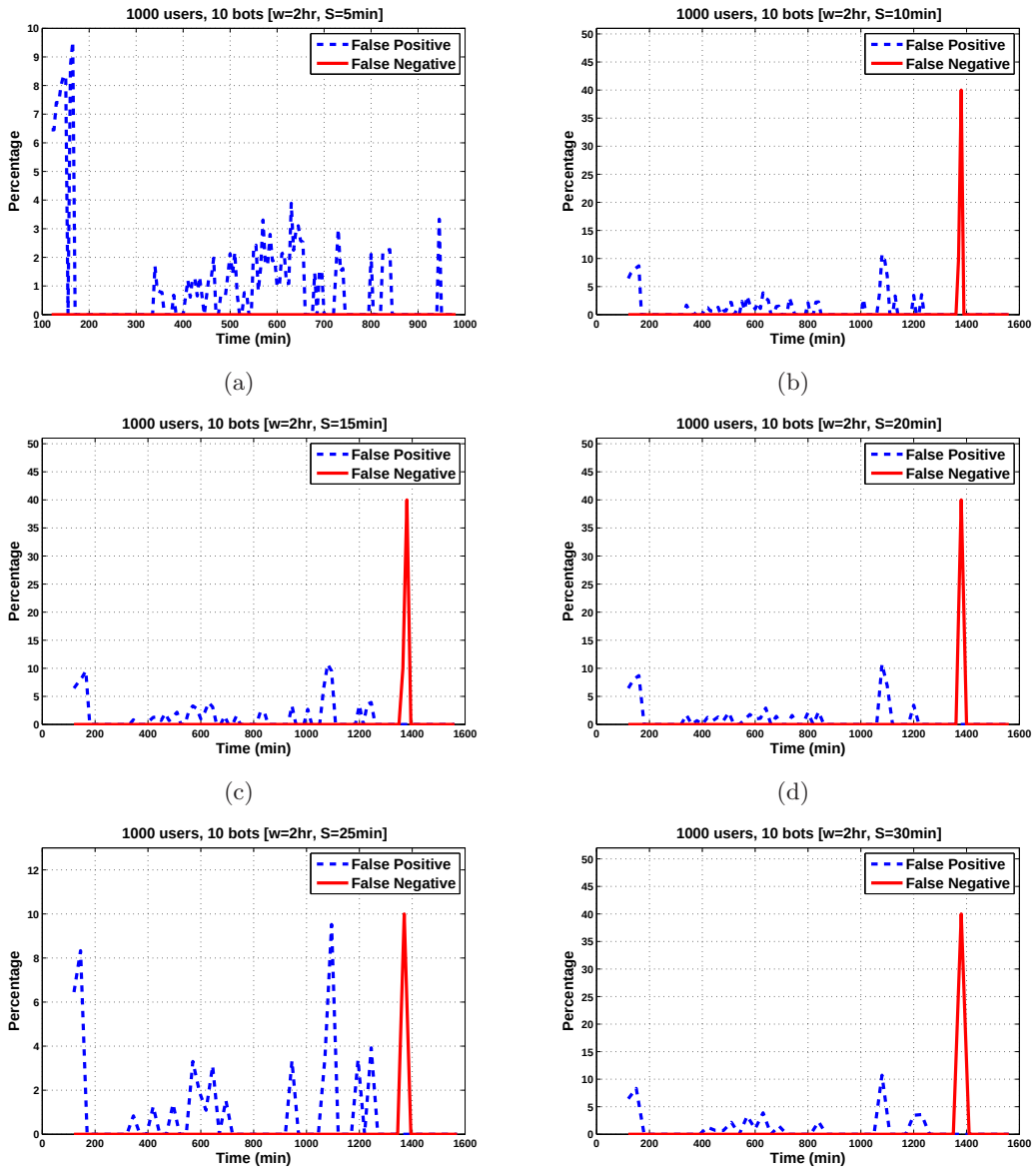


Figure 4.6: Set-B Experiment Results at W= 2hr & Bots= 10: For 6 sliding win sizes. Maximum FP is 10.8%, and FN is almost 0%. FP/FN values were not directly impacted by different sliding win sizes.

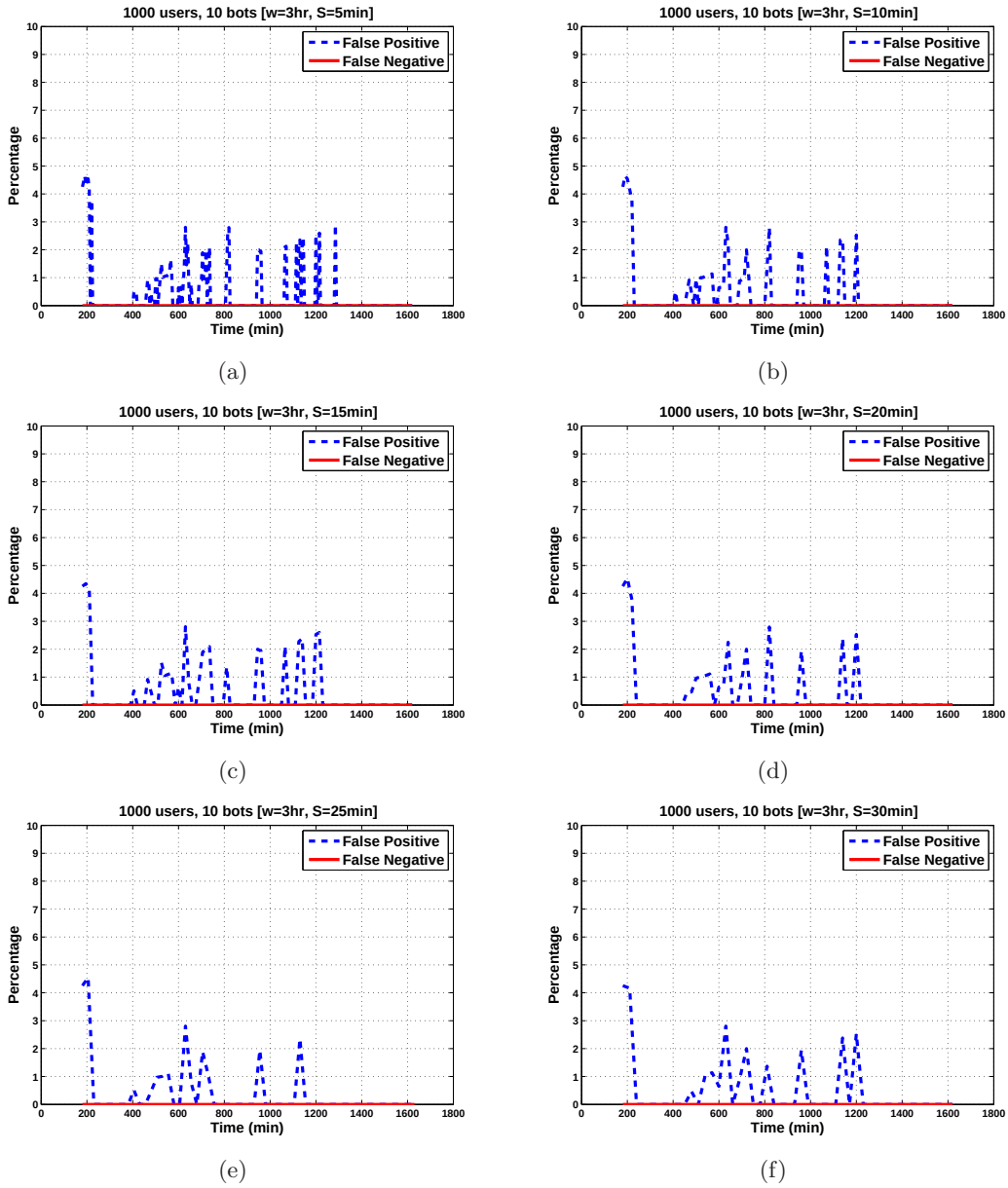


Figure 4.7: Set-B Experiment Results at W= 3hr & Bots= 10: For 6 sliding win sizes. Maximum FP= 4.7%, and FN= 0%. The best FP values were at S= 15m and 30m, although the difference is small. The maximum difference between the maximum FP across all sliding window sizes is 0.05%.

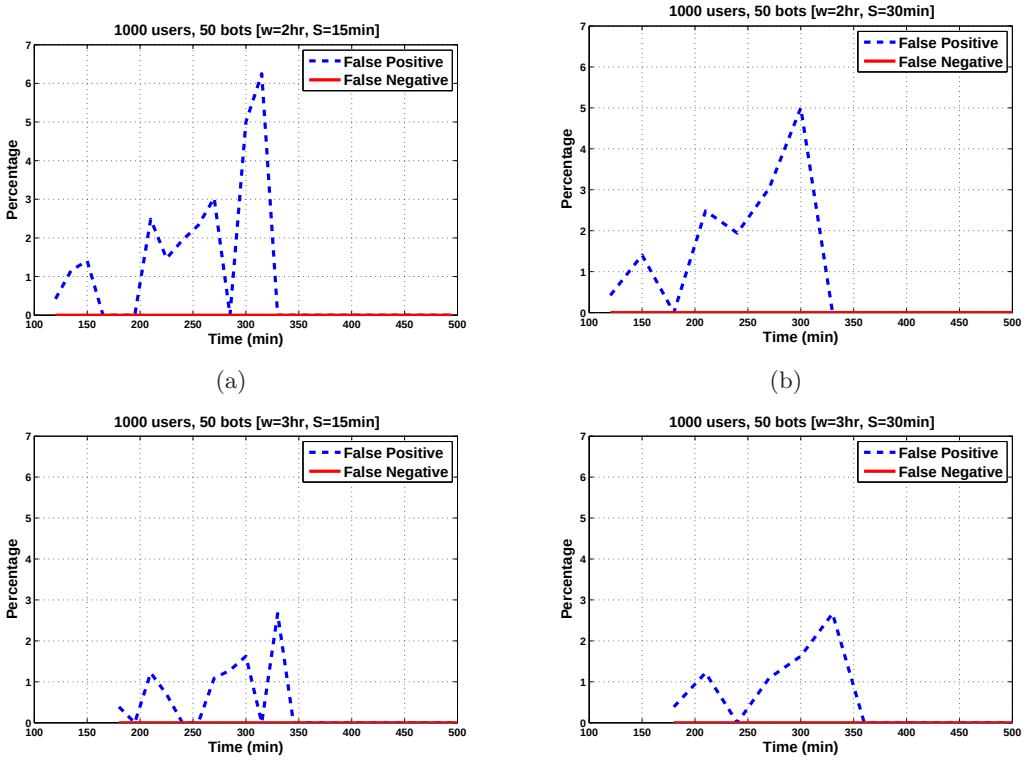


Figure 4.8: Set-C Experiment Results: Maximum FP/3hr= 2.7%, Maximum FP/2hr= 6.3%, and FN= 0%.

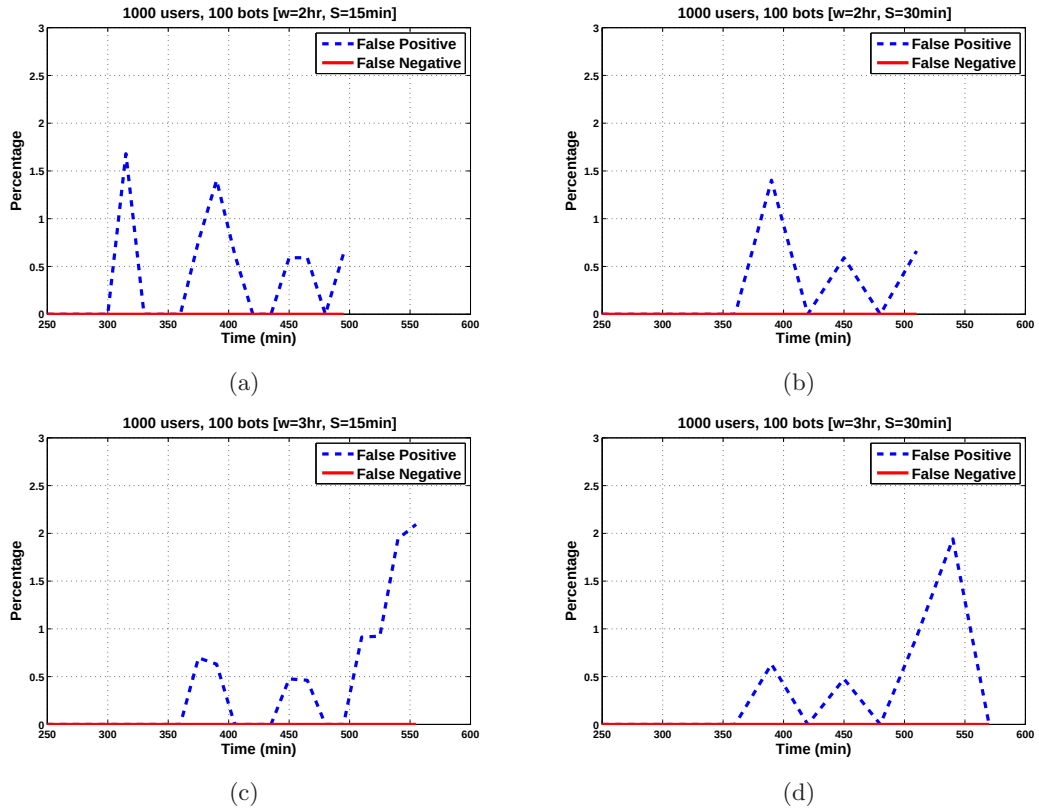


Figure 4.9: Set-D Experiment Results: Maximum FP/2hr= 1.7%, Maximum FP/3hr= 2.1%, and FN= 0%.

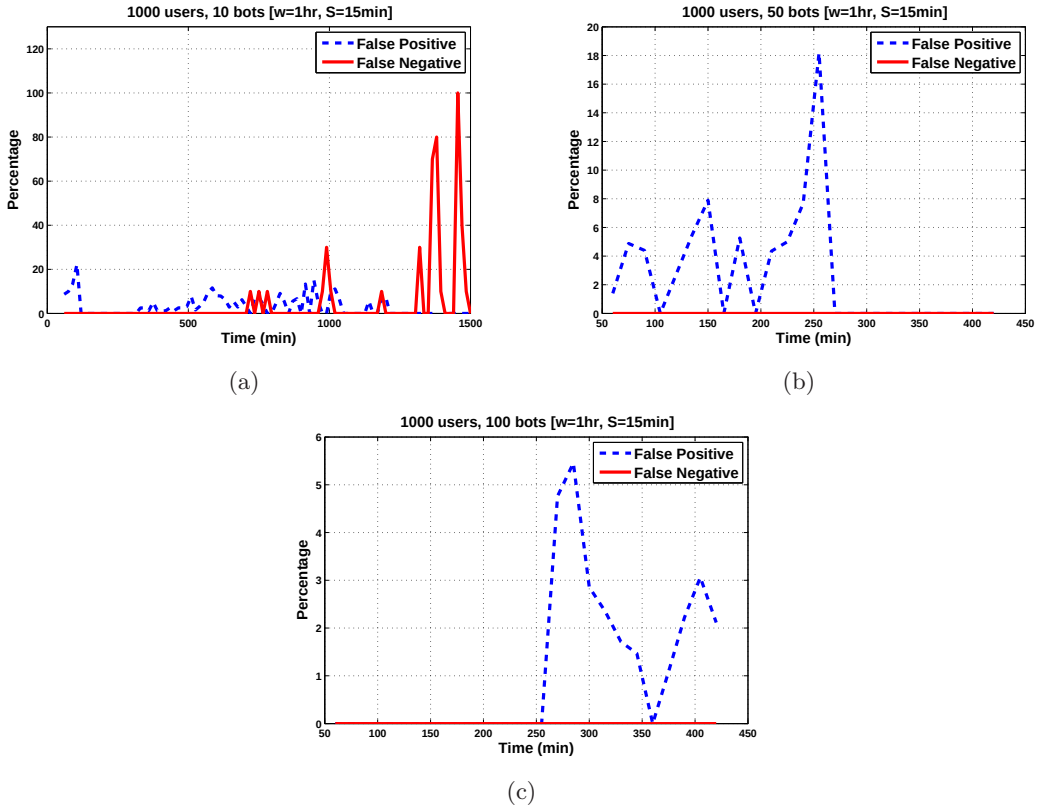


Figure 4.10: Set-E Experiments Results: Maximum FP/10bots= 22%, Maximum FP/50bots= 18%. FN/50,100bots= 0% and unstable FN at Bots= 10.

Chapter 5

Conclusions and Future Work

5.1 Conclusions

Botnets are a serious threat for the Internet, because of the huge number of potential bots and the tremendous resources that they can provide for the remote attackers. The remote attacker controls a botnet over a Command and Control channel. Botnets are deployed over different kinds of protocols and come in centralized and distributed forms.

The Session Initiation Protocol (SIP) is attracting more attention and wide adoption. In practice SIP provides many options and features that can be exploited. The SIP infrastructure minimizes management overhead. Therefore, SIP can be a useful Command and Control channel protocol for botnets.

We proposed a novel approach to detect SIP botnets. Our approach looks for similar behaviors in the Command and Control communications. We measured the similarity among users using the Jaccard similarity coefficient. We implemented an online detection system based on that. The proposed system is independent from the message contents, and the sequence of events. The proposed system consists of two main components: Monitoring engine and Correlation & Detection engine.

We conducted several sets of experiments to rigorously evaluate the system. We tuned the two main thresholds in the botnet detection system: α and β. α is a threshold that helps measuring the similarity between two SIP sessions, and β is a threshold that helps measuring the similarity between two users. We found that the best value of α is 0.05 and the best value of β is 0.8. We evaluated the precision of the botnet detection system by calculating the false positive and false negative rates in different settings. We found that

the proposed system achieves the lowest false positive and false negative when we apply the detection algorithm over a window of 3 hours of SIP sessions.

5.2 Future Work

There are multiple areas of potential improvements for the proposed system in this project. The first area is to improve the time complexity of calculating the similarity between two users from $O(M^2)$ into $O(M)$ which requires a redesigning of the Correlation and Detection engine. We attempted to do that by maintaining a sorted list of user sessions based on their length. We computed the similarity between two users in $O(N)$ time by trying 3 types of similarity/distance algorithms: the Cosine Similarity, the Canberra distance, and the Normalized Euclidean distance. At this time we calculated the difference between two users in different way. Let us define a feature vector as a vector that represents all values of a feature across all sessions, then let us assume that A is a feature vector for user U_1 and B is a feature vector for user U_2. Therefore, the distance between U_1, U_2 on that specific feature is measured by the distance between A and B. The general distance between U_1, U_2 is measured by taking the average distance of all feature vector distances. The reason behind maintaining a sorted list of sessions based on their length is to eliminate the session time factor. Unfortunately, we could not find the right threshold settings that minimizes both false positive and false negative. We have two challenges. The first one is to normalize the number of sessions for each user to have an equal size of feature vectors among all users. The second one is to improve the sort technique in order to have a better chance that each closest feature value pair in two feature vectors A and B will face each other, which would provide us with a more accurate distance between A and B.

Another area of improvement is to transfer the system to be a protocol independent detection system by tracking IP flows instead of specifically tracking SIP sessions that requires some SIP header values.

A third improvement area is handling more evasion techniques, such as using pool of random SIP options, generating random session lengths, or generating random noise packets. These evasion techniques try to make bots look and behave differently, while still trying to achieve various malicious activities. They worth more study and analysis although implementing them is definitely not simple and requires complex programming and networking

skills. The management overhead will extremely increase accordingly. We will briefly discuss the above evasion techniques in the following. Implementing those evasion techniques will not necessarily result in bypassing the proposed detection system. For instance, the proposed system does not rely on a specific SIP option, which means that applying different SIP options is simply not enough to affect its performance. Applying different SIP options such as using message header, message body, or user-defined header does not necessarily result in a large distance on session's feature vector values. From another perspective, such evasion techniques might have negative impacts on the botnet functionality as well. For example, generating random SIP session lengths is going to slowdown the propagation of botmaster instructions, and receiving the feed back. Slowing down the propagation of instructions and feedback has a negative impact on the botnet efficiency. Another example, is generating random noises which can be noticeable. Noise traffic can create abnormal network patterns that might draw the attention of the anomaly based detection system that looks for abnormal behaviors.

Bibliography

[1] P. Wang, B. Aslam, and C. Zou. Peer-to-peer botnets. In *Handbook of Information and Communication Security*, volume 25. Springer, February 2010.

[2] J. Zhuge, T. Holz, X. Han, J. Guo, and W. Zou. Characterizing the IRC-based botnet phenomenon. Technical report, Peking University & University of Mannheim Technical Report, December 2007.

[3] M. Collins, T. Shimeall, S. Faber, J. Janies, R. Weaver, M. Shon, and J. Kadane. Using uncleanliness to predict future botnet addresses. In *Proc. of Internet Measurement Conference (IMC'07)*, pages 93 – 104, San Diego, CA, October 2007.

[4] Z. Zhu, G. Lu, and Y. Chen. Botnet research survey. In *Proc. of IEEE International Computer Software and Applications Conference (COMPSAC'08)*, pages 967–972, Turku, Finland, July 2008.

[5] W. Strayer, D. Lapsely, R. Walsh, and C. Livadas. Botnet detection based on network behavior. In *Botnet Detection*, volume 36 of *Advances in Information Security*. Springer, October 2007.

[6] G. Gu, J. Zhang, and W. Lee. Botsniffer: Detecting botnet command and control channels in network traffic. In *Proc. of Network and Distributed System Security Symposium (NDSS'08)*, San Diego, CA, February 2008.

[7] X. Yu, X. Dong, G. Yu, Y. Qin, D. Yue, and Y. Zhao. Online botnet detection based on incremental discrete fourier transform. *Journal of Networks*, 5(5):568–576, May 2010.

[8] The 3rd Generation Partnership Project (3GPP). http://www.3gpp.org/article/ims.

[9] M. Bailey, E. Cooke, F. Jahanian, Y. Xu, and M. Karir. A survey of botnet technology and defenses. In *Proc. of Homeland Security Conference (CATCH'09)*, pages 299–304, Washington D.C., March 2009.

[10] D. Dagon, G. Gu, and C. Lee. A taxonomy of botnet structures. In *Botnet Detection*, volume 36 of *Advances in Information Security*. Springer, October 2007.

[11] J. Bambenek and A. Klus. Botnets and proactive system defense. In *Botnet Detection*, volume 36 of *Advances in Information Security*. Springer, October 2007.

[12] E. Cooke and F. Jahanian. The zombie roundup: Understanding, detecting, and disrupting botnets. In *Proc. of Steps to Reducing Unwanted Traffic on the Internet Workshop (SRUTI'05)*, pages 39–44, Cambridge, MA, July 2005.

[13] Z. Li, A. Goyal, and Y. Chen. Honeynet-based botnet scan traffic analysis. In *Botnet Detection*, volume 36 of *Advances in Information Security*. Springer, October 2007.

[14] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session initiation protocol. RFC 3261, June 2002.

[15] M. Garcia-Martin. Input 3rd-Generation Partnership Project (3GPP), release 5 requirements on the session initiation protocol (SIP). RFC 4083, May 2005.

[16] T. Berners-Lee, R. Fielding, U.C. Irvine, and L. Masinter. Uniform resource identifiers (URI): Generic syntax. RFC 2396, August 1998.

[17] M. Handley and V. Jacobson. SDP: Session description protocol. RFC 2327, April 1998.

[18] A. Berger and M. Hefeeda. Exploiting SIP for botnet communication. In *Proc. of Secure Network Protocols Workshop (NPSec'09)*, Princeton, NJ, October 2009.

[19] J. Rosenberg. A presence event package for the session initiation protocol (SIP). RFC 3856, August 2004.

[20] B. Campbell, J. Rosenberg, H. Schulzrinne, C. Huitema, and D. Gurle. Session initiation protocol (SIP) extension for instant messaging. RFC 3428, December 2002.

[21] H. Zeidanloo and A. Abdul Manaf. Botnet detection by monitoring similar communication patterns. *International Journal of Computer Science and Information Security*, 7(3):36–45, March 2010.

[22] G. Gu, R. Perdisci, J. Zhang, and W. Lee. Botminer: Clustering analysis of network traffic for protocol- and structure-independent botnet detection. In *Proc. of USENIX Security Symposium (Security'08)*, San Jose, CA, July 2008.

[23] Snort: Open source network intrusion prevention and detection system. http://www.snort.org/.

[24] G. Boggia, P. Camarda, A. D'Alconzo, A. Biasi, and M. Siviero. Drop call probability in established cellular networks: from data analysis to modelling. In *Proc. of the Vehicular Technology Conference (VTC'05)*, pages 2775 – 2779, Stockholm, Sweden, May 2005.

[25] T. Holz, M. Steiner, F. Dahl, E. Biersack, and F. Freiling. Measurements and mitigation of peer-to-peer-based botnets: A case study on storm worm. In *Proc. of USENIX Workshop on Large-Scale Exploits and Emergent Ehreats: Botnets, Spyware, Worms, and more (LeeT'08)*, San Francisco, CA, April 2008.

[26] Opensips: Open source implementation of SIP server. http://opensips.org/.