Czech Technical University in Prague
Faculty of Electrical Engineering
Department of Computer Science

Master's Thesis

Behavioral Analysis and Detection of IoT malware using the IRC protocol

Bc. Ondřej Preněk

Supervisor: Ing. Sebastián García, Ph.D.

Study Programme: Open Informatics

Field of Study: Artificial Intelligence

August 21, 2020

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Preněk Ondřej**	Personal ID number:	**460532**
Faculty / Institute:	**Faculty of Electrical Engineering**		
Department / Institute:	**Department of Computer Science**		
Study program:	**Open Informatics**		
Specialisation:	**Artificial Intelligence**		

II. Master's thesis details

Master's thesis title in English:

Behavioral Analysis and detection of IoT malware using the IRC protocol

Master's thesis title in Czech:

Analýza chování a detekce IoT malwaru používající protokol IRC

Guidelines:

This thesis aims to solve the problem of identification and classification of botnets using the IRC protocol. In the last years, IRC has been used again as the main Command and Control protocol for Iot botnets. IRC is an old and well known protocol, but it has not been studied for IoT malware. The study of IRC is complex since it can work as a centralized protocol, or a peer-to-peer protocol. The goal of the thesis is to analyse malicious IRC communication and normal IRC communications in order to learn how to classify them. We propose a technique for detecting malicious IRC communications in the network
1) Monitor individual botnet families and understand their behavior.
2) Analyse and compare botnets' behavior.
3) Design, implement and experimentally verify a technique for detecting and classifying malicious IRC communication in the network

Bibliography / sources:

[1] CAGLAYAN, Alper, et al. Behavioral analysis of botnets for threat intelligence. Information systems and e-business management, 2012, 10.4: 491-519.
[2] FEILY, Maryam; SHAHRESTANI, Alireza; RAMADASS, Sureswaran. A survey of botnet and botnet detection. In: 2009 Third International Conference on Emerging Security Information, Systems and Technologies. IEEE, 2009. p. 268-273.

Name and workplace of master's thesis supervisor:

Ing. Sebastián García, Ph.D., Artificial Intelligence Center, FEE

Name and workplace of second master's thesis supervisor or consultant:

Date of master's thesis assignment: **04.02.2020** Deadline for master's thesis submission: __________

Assignment valid until: **30.09.2021**

[illegible]	[illegible]	[illegible]
Ing. Sebastián García, Ph.D.	Head of department's signature	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature		Dean's signature

III. Assignment receipt

The student acknowledges that the bachelor's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the bachelor's thesis, the author must state the names of consultants and include a list of references.

.

Date of assignment receipt	Student's signature

Aknowledgements

Foremost, I would like to express my gratitude to the Stratosphere team for their support throughout the whole project period. Their guidance helped me in all times of research and writing of this thesis. My sincere thanks also go to my advisor Sebastian Garcia. Besides guidance during the research, he provided me with an opportunity to join their team. I thank my university classmates for all the fun we have had in the last five years. Last but not least, I would like to thank my family and my girlfriend for providing me with unfailing support and continuous encouragement throughout my years of study and writing the thesis.

Declaration

Abstract

The security of IoT devices represents a severe concern to millions of people. IoT devices are very hard to defend, and very easy to attack, usually ending up as part of a botnet network that in turn attacks other devices around the world. In the past years, the IRC protocol has been re-discovered as the main Command and Control protocol for IoT botnets. IRC is an old and well-known protocol, but it has not been studied yet in terms of IoT malware. This thesis studies the problem of identification and classification of IoT botnets that use the IRC protocol.

Our proposal to detect IRC botnet traffic in a network has two main parts. First, we present a machine learning detection based on what we call 3-tuple IRC sessions. An IRC 3-tuple session is defined as the group of flows between a source IP address, a destination IP address, and a destination port. 3-tuples allow us to gather and identify all the flows related to the same IRC session and server. Our machine learning model obtained 95, 24% F1-score in this task. Second, we propose a technique that analyzes the characteristics of IRC channels and IRC users, and applies machine learning for detection. This technique studies certain features of IRC channels, such as entropy of the messages, and features of IRC users, such as diversity of their messages, in order to find malicious IRC traffic. Our machine learning model on IRC channels obtained 80% F1-score and the one on IRC users obtained 66.67% F1-score. A new dataset was used (designed and created by the Aposemat Project) that includes real IRC traffic from malware executions and benign IRC traffic from a large University. Finally, a comparison against the state-of-the-art Intrusion Detection Systems Snort and Suricata was done in order to verify that our technique was sound. All the code of the detectors was published as a new public Zeek module and as a module for the Slips system (Stratosphere Linux IPS).

Key words: IRC, Botnet, IoT, Malware, Machine Learning

Abstrakt

Bezpečnost IoT zařízení představuje vážnou hrozbu pro miliony lidí. IoT zařízení je velmi obtížné chránit a obvykle končí jako součást Botnet sítě, která následně útočí na jiná zařízení na světě. V posledních letech byl IRC protokol znovu objeven jako hlavní command-and-control protokol pro IoT botnety. IRC je starý a dobře známý protokol, ale nebyl zkoumán v souvislosti s IoT malwarem. Tato práce si klade za cíl studovat problém identifikace a klasifikace IoT botnetů, které používají protokol IRC.

Náš návrh detekovat škodlivý IRC provoz v síti má dvě části. Nejprve představíme detekci pomocí technik strojového učení na základě toho, čemu říkáme IRC třičlenné n-tice. IRC třičlenná n-tice je definována jako skupina toků mezi shodnou zdrojovou IP adresou, cílovou IP adresou a cílovým portem. Výsledný model strojového učení dosáhl F1-skóre 95.24%. V druhé části práce navrhujeme techniku, která analyzuje vlastnosti IRC kanálů a IRC uživatelů a aplikuje strojové učení pro detekci. Tato technika studuje vlastnosti IRC kanálů, jako je entropie zpráv, a vlastnosti IRC uživatelů, jako je rozmanitost jejich zpráv, za účelem nalezení škodlivého přenosu IRC. Náš model strojového učení na detekci škodlivých IRC kanálů dosáhl F1-skóre 80.00% a model na detekci škodlivých IRC uživatelů vykázal F1-skóre 66.67%. Byl navržen a vytvořen nový dataset tak, aby zahrnoval skutečný IRC provoz z IoT zařízení infikovaných malwarem a neškodlivý IRC provoz získaný z velké univerzity. Nakonec bylo provedeno srovnání s nejmodernějšími IDS Snort a Suricata, aby se ověřilo, že naše technika byla správná. Výsledná práce byla publikována pro veřejnost jako nový Zeek modul a jako modul pro systém Slips (Stratosphere Linux IPS).

Klíčová Slova: IRC, Botnet, IoT, Malware, Strojové učení

Překlad titulu: Analýza chování a detekce IoT malwaru používajícího protokol IRC

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The current massive attacks on the Internet are mostly driven by the large volume of IoT devices that are being attacked and infected. This is mainly due to the high number of vulnerabilities and the increasing adoption of IoT. IoT device vulnerabilities give attackers an easy way to access the network so they can target more critical devices or systems. These infected IoT devices are difficult to detect, leading to many unprotected networks. In 2019, the F-Secure company measured a three-fold increase in attack traffic to more than 2.9 billion events [29].

Once an IoT device is attacked, its resources are accessed and its actions are controlled by malware. This malware is usually controlled by means of a command and control channel. A command and control channel is a way in which the attacker can control the devices remotely and simultaneously. A group of infected devices controlled by a command and control are called a botnet [25]. A botnet consists of bots (infected devices) and one or more Command and Control (C&C) servers which control bots by sending commands to them. Bots are highly customizable so they can be used to perform various types of tasks. Botnets can be used to perform distributed denial-of-service attacks (DDoS attack), steal data, send spam, and allow the attacker to access the device without the owner permission.

A common command and control channel protocol used for IoT botnets has been the Internet Relay Chat (IRC) protocol. IRC is an old and well-known protocol used to chat between users which has been also used a lot in the first generations of traditional botnets. In the last years we have witnessed a growth in the use of IRC for IoT botnets. Despite this grow, IRC has not been studied in terms of IoT malware. The traditional IRC protocol is designed to group communication in channels, and therefore it represents a suitable communication resource in order to send the commands from the C&C server to the bots.

The problem of botnet detection was studied by other authors in the past. Livadas et al. focused on identifying IRC-based botnet traffic using conditional probability methods, achieving both a low false-negative score of 2.49% and a false-positive score of 15.04% [24]. Claudio Mazzariello introduced a network traffic analysis architecture and described a behavioral model for detecting IRC botnets [28]. This behavioral model identified botnet-related activities based on the behavior of the IRC user in a channel with an accuracy of 100%. However, further investigation of the results is needed in order to understand whether the obtained results were biased or not.

This thesis studies the problem of identifying and classifying IoT botnets using the IRC protocol by means of machine learning techniques based on behavioral features. In this thesis we created a method to detect malicious IRC communications in a network. We hope that this research can help secure IoT devices from being controlled as part of a botnet.

There are two main parts of this thesis. In the first part, we present a new technique for detecting malicious IRC 3-tuple sessions in the network. An IRC 3-tuple session is defined as the group of flows between a source IP, a destination IP, and destination port. The idea of the 3-tuple is to gather and identify all the flows related to the same IRC session. In the second part, we present a new technique for detecting malicious IRC channels and IRC users. To train and test the algorithms, we used a new dataset, created by the Aposemat project [11]. The dataset contains both malicious traffic from IoT botnets using the IRC protocol, and normal traffic from IRC networks. This dataset provided us with the features needed to create our models.

The results from the experiments obtained an F1-score of 96.24% for detecting IRC 3-tuples sessions, an F1-score of 80% for detecting malicious IRC channels, and an F1-score of 66.67% for detecting malicious IRC users. These results are significant and provide a good indication that detecting IRC malware is possible, but detecting malicious IRC channels and malicious IRC users can be challenging. A comparison of our detection methods with the state-of-the-art IDS tools Snort and Suricata verified that our technique is sound and can outperformed the state-of-the-art techniques.

This thesis is organized as follows. Chapter 2 describes the IRC protocol and the botnet phenomenon. Chapter 3 describes related work in the field of botnet detection, Chapter 4 explains how and what data was collected, the methodology for capturing malware traffic, and finally how the data is visualized. In Chapter 5, a new technique for detecting malicious IRC 3-tuple sessions in the network is proposed. In Chapter 6, a new technique of detecting malicious IRC channels and IRC users is proposed. Chapter 7 compares the efficiency of proposed methods with the state-of-the-art Intrusion Detection Systems (IDS). Finally, in Chapter 8, the work is concluded, and future work is discussed.

Chapter 2

Background

To understand the basic concepts of this thesis, this chapter describes the background terms and techniques. First, we will describe the IRC protocol and its design as well as the biggest IRC networks, both in the past and in the present. Then, we will describe what is a botnet, its architecture, its intended usage, and the most well known IoT botnets that use the IRC protocol.

2.1 Internet Relay Chat

Internet Relay Chat (IRC) is a text-based application layer protocol that allows users to exchange messages in real-time. IRC is mainly designed for group communication in channels [31]. It follows a client/server networking model, and consists of different, separate networks. A client is any device connecting to an IRC server that is not another server. Each client is distinguished from other clients by a unique nickname [34]. There are a lot of available client implementations, such as mIRC, HexChat, and IRSSI, as well as a lot of server implementations, such as the original IRCd. The servers provide a point to which clients may connect and talk to each other [36]. Moreover, the server provides a point for other servers to connect to, forming an IRC network.

The IRC network is configured in such a way that the network forms a spanning tree where each server acts as a central node for the rest of the network [33]. Messages are routed only through necessary tree branches, while the state of the network is sent to every server. The sample architecture of IRC network is depicted in Figure 2.1.

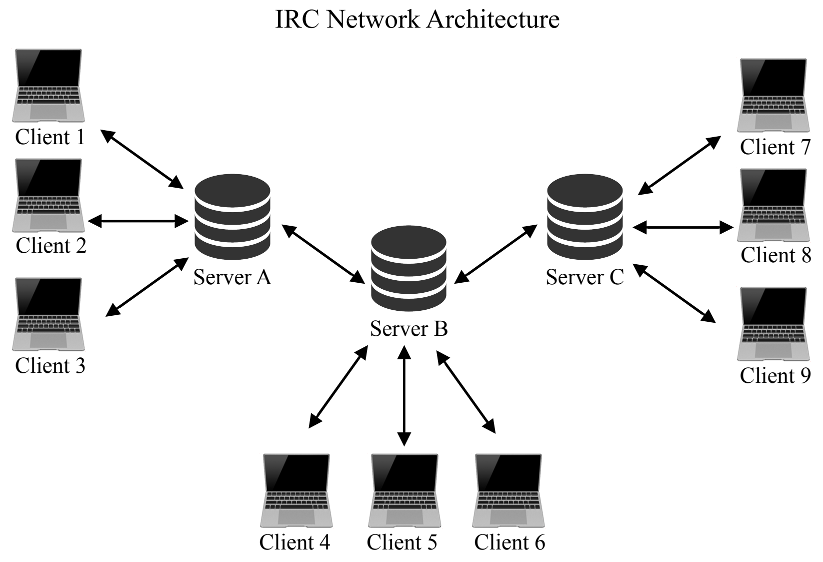

Figure 2.1: Example of a small IRC network

There are three classes of communication that are used in the IRC network: one-to-one, one-to-many, and one-to-all. The one-to-one communication is mostly performed by clients. The path of a delivering message is the shortest path between any two points in the IRC network. The following examples of possible message delivery paths are references to the IRC network architecture illustrated in Figure 2.1:

- A message between Client 1 and Client 2 is only seen by Server A, which sends it directly to Client 2.
- A message between Client 1 and Client 4 is seen by Server A, Server B, and Client 4. No other clients or servers are allowed to see the message.
- A message between Client 1 and Client 7 is seen by Server A, Server B, Server C and Client 7.

The one-to-many communication is the main focus of IRC. Depending on the purpose, IRC uses several types of one-to-many communication:

- **To a list** - To send a message to a list of users, the server breaks the list up and dispatches a separate copy of the message to each destination. The server sends all message copies without checking whether they do not follow duplicated network path multiple times.
- **To a group (channel)** - The channel conversation is only sent to servers which are supporting users on the given channel. If multiple users of the given channel are on the same server, the message is sent only once to that server, and then the message is sent to each client on the channel.

- **To a host/server mask** - To send a message to a large community of users, host and server mask messages are created. These messages are sent only to users whose host or server information match that mask.

The one-to-all communication is performed to broadcast message to all clients or servers or both. For some messages, it is needed to broadcast it to all servers so that the state information held by each server is consistent in the IRC network.

This mean of communication that IRC follows has a several issues. A broken or malicious server can cause major damage to the network [32]. All network structure changes, intentional or unintentional, require a netsplit and netjoin [35]. This causes a lot of unnecessary network traffic, false join/quit messages to users, and temporary loss of user communication.

2.1.1 IRC Commands

IRC commands allow a user to perform specific functions on IRC, such as maintaining control of a channel and the users on it. If the message starts with a "/" character, it will be interpreted as an IRC command. Although some commands may be supported by specific IRC client programs, this is a list of the most common commands:

- /**HELP** - shows general help or help for the command given as an argument
- /**MSG** - sends a private message to a specified user
- /**LIST** - lists all channels
- /**INVITE** - invites another user to a channel
- /**JOIN** - joins a channel
- /**TOPIC** - changes a channel topic
- /**NICK** - changes a nickname
- /**WHOIS** - shows information about user
- /**KICK** - removes the user from the channel
- /**LEAVE** - leaves a channel
- /**QUIT** - exits current IRC session
- /**AWAY** - leaves a message regarding user unavailability

2.1.2 Popularity of IRC

The highest usage of IRC was recorded in 2003 where the biggest IRC network, QuakeNet, reached about 200,000 daily users [2]. QuakeNet focuses on an online gaming community, but it also consists of communities covering most diverse topics [1]. The number of daily users decreased as time passed as is depicted in Figure 2.2, which shows the top 10 IRC networks in 2019 in terms of number of users. There are several reasons for this drop in usage. The most important of which are the expansion of social networks and easily accessible messaging services for a wider range of people, such as Whatsapp, Slack, Telegram, etc. In 2019, the current biggest IRC network, Freenode, has about 90,000 users [3]. Besides the high popularity of IRC chats between users, IRC is also frequently used in botnets as a communication protocol.

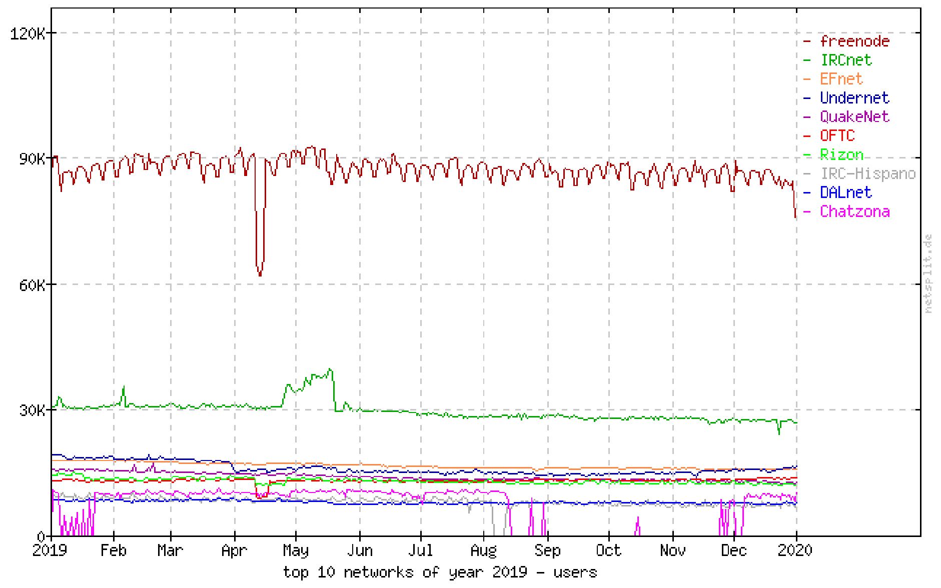

Figure 2.2: Top 10 IRC Networks of 2019 by number of users [3]

2.2 Botnet

A "bot network", or "botnet", is defined as a concerted network of bots capable of acting on instructions generated remotely [25]. The botnet controller, referred to as botmaster, uses one or more Command and Control (C&C) servers to control infected devices, called bots. A botmaster controls bots by sending commands to them. In order to provide a better understanding of the botnet phenomenon, this section elaborates on botnet structure, botnet life cycle, botnet architecture, and usage of IRC in botnets.

A botnet is composed of three parts: bot, botmaster, and C&C server(s). The bot, also known as a zombie, is a type of malware installed in a compromised computer. The bots are remotely controlled by the botmaster's commands sent via the C&C server. A botmaster is a person who uses the C&C server(s) for remote bot control. Botmasters can, and often do, disguise their IP addresses in order to avoid the detection of investigators and law enforcement by using proxies or TOR. Sometimes, a botnet is controlled by multiple botmasters who operate together. A C&C server is a server controlled by a botmaster used for sending commands to bots and receiving stolen data from infected devices.

2.2.1 Botnet Life Cycle

The botnet life cycle consists of five phases: the initial infection, the secondary injection, the connection, the communication with the malicious C&C, and the maintenance & update. This life cycle is depicted in Figure 2.3, where all the steps of the connections and communications are illustrated. The phases can be described as follows:

1. **Initial Infection Phase**

   In the first phase, the attacker scans a target sub-network for known vulnerabilities and infects the targeted device through various exploitation methods.

2. **Secondary Injection Phase**

   In the second phase, the infected host executes a script that fetches the image of the actual bot binary. The bot binary is installed on the targeted device, and once installed, the victim computer is converted into a zombie and runs the malicious code.

3. **Connection Phase**

   In the connection phase, the bot program establishes a connection with the C&C server, and it becomes a part of the attacker's botnet. This connection can be established by using different protocols such as HTTP, HTTPs, IRC, ect (add ref)

4. **Attack Phase**

   After the connection phase, C&C activities of botnet begins. The botmaster uses the C&C channel to distribute commands to bots for their execution. The C&C server enables the botmaster to remotely control the action of large number of bots to conduct various illicit activities.

5. **Update & Maintenance Phase**

   The last phase is dedicated to keeping bots alive and up to date. In this phase, bots receive a command to download an updated binary. Bot controllers may need to update the bots for several reasons, such as binary updates to prevent detection, or addition of new features. Besides that, sometimes the binary is updated due to moving the bot to a different C&C server. This process is called server migration and is done in order to keep the botnet alive [14]. Using Dynamic DNS (DDNS), the botmaster tries to keep the botnet invisible and portable. DDNS is a resolution service that ensures frequent updates and changes in server location [37]. In case of blacklisting a C&C server at a certain IP address, the botmaster can easily set up another C&C server at a different IP address. After that, bots migrate to the new C&C server and stay alive.

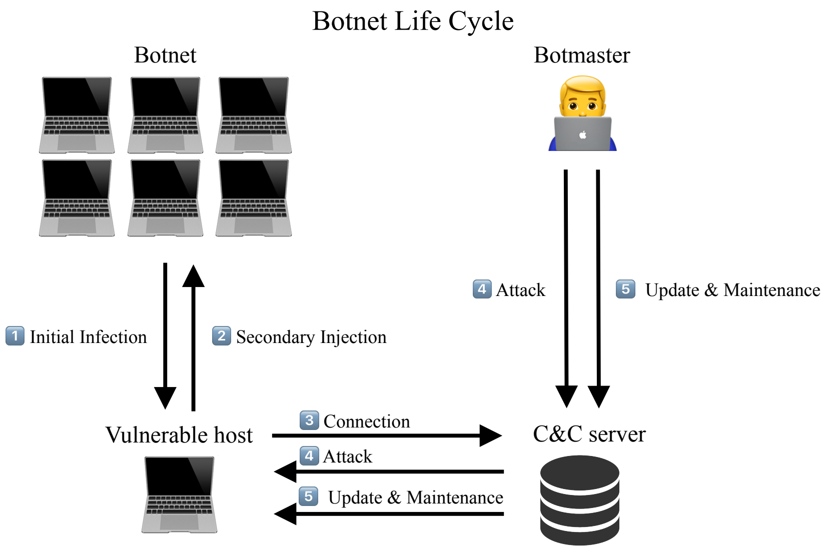

Figure 2.3: Botnet Life Cycle

2.2.2 Botnet Architecture

The architecture of a botnet defines a connection method between the C&C server and bots. There are currently three botnet architectures: centralized, decentralized, and hybrid.

In a centralized architecture, all the bots are controlled by a central point, which makes C&C server visible for detection and termination. The most common client-server communication protocols are IRC or HTTP. Figure 2.4 shows an example of botnet with centralized architecture.

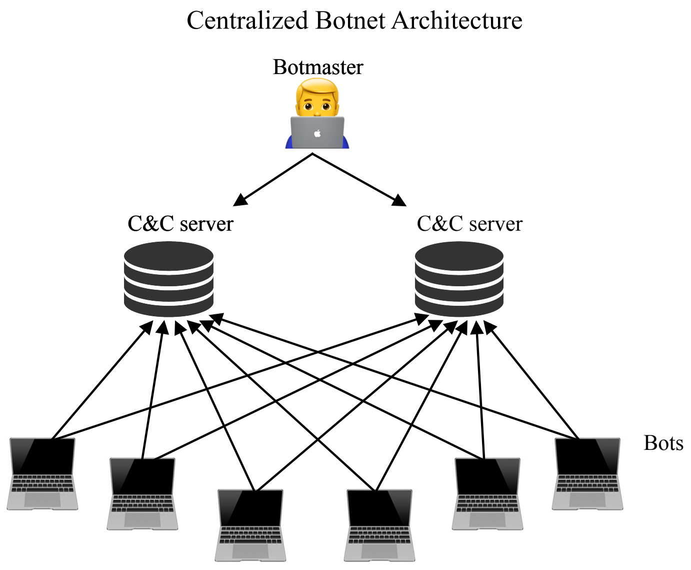

Figure 2.4: Centralized Architecture of Botnet

In a decentralized architecture, the C&C server is not specified. This type of botnet works in Peer-to-Peer (P2P) mode where each bot registers available bots in its surroundings. Each bot continuously monitors new bots and updates state information with other bots. Due to the use of dynamic routing methods to create and operate such a network, a decentralized botnet is more challenging to set up compared to a centralized botnet. However, a decentralized botnet is more difficult to detect, be monitored and blocked by law enforcement authorities. An example of a botnet with a decentralized architecture is depicted in Figure 2.5.

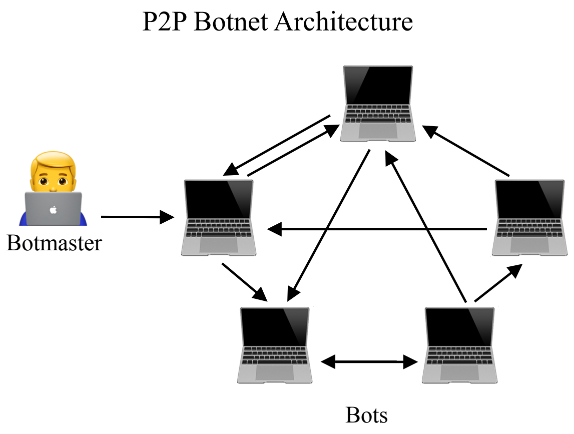

Figure 2.5: Decentralized Architecture of a Botnet

A hybrid architecture incorporates the advantages of centralized and decentralized architectures by having a structure of decentralized IRC networks where bots connect to the C&C server in the way of a typical client-server model. In this architecture, the risk of detection of the entire botnet is significantly reduced, because if a C&C server gets detected, only a subset of bots would be identified. An example of a botnet with the hybrid architecture is depicted in Figure 2.6.

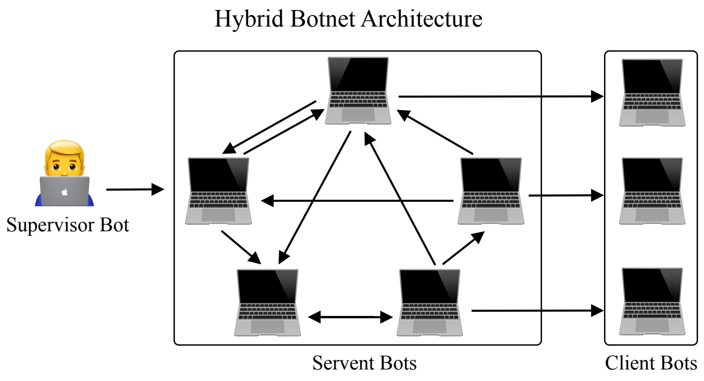

Figure 2.6: Hybrid Architecture of a Botnet

2.2.3 IoT Botnets

There is a wide variety of IoT botnets. One of the well known IRC-based IoT botnets is called Mirai. It consists of approximately 500,000 compromised IoT devices with a traffic peak of 1.2 Terabits per second [40, 17]. Compared to its predecessor, Bashlite, Mirai uses more resilient hosting and control infrastructures and supports more effective attacks [26]. In addition, the release of Mirai's source code allowed malware creators to customize and refine the code, allowing several variants of malware to emerge, such as Persirai, BrickerBot, or HideNSeek [23].

A timeline of when IoT botnets appeared can be seen in Figure 2.7, which shows how IoT botnets have evolved in the last years. The timeline depicts when individual botnet families were created, when the source code of a botnet family leaked online, and how long a botnet family remained active. Figure 2.8 shows a timeline of protocols that were used in particular botnets. It is evident from the timeline that the majority of botnets use P2P, IRC or a custom protocol.

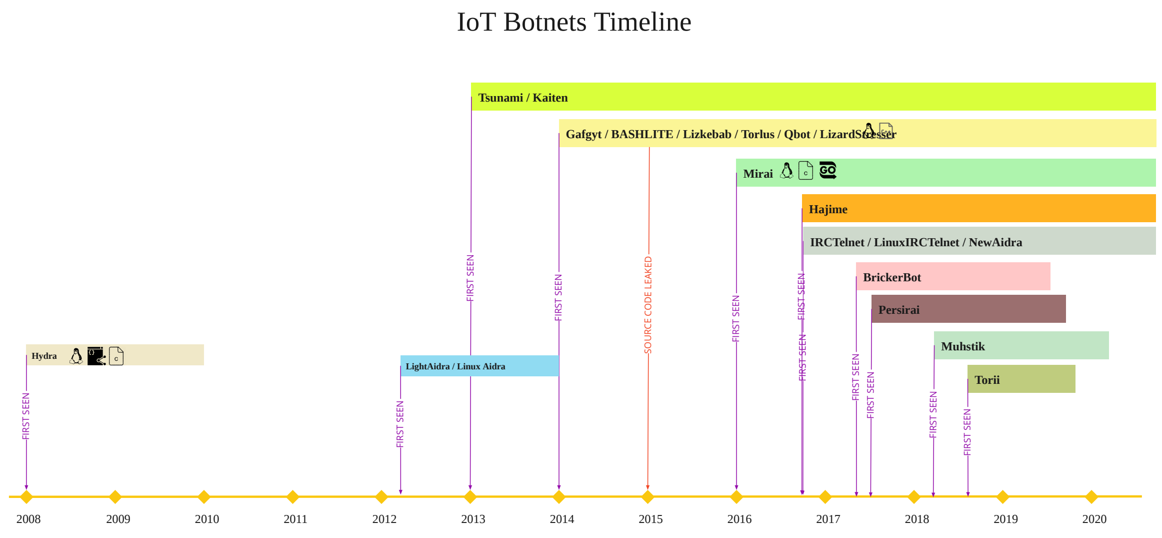

Figure 2.7: Timeline of IoT botnet families and an estimation of when they appeared. (Author Veronica Valeros, reproduced with permission) [45]

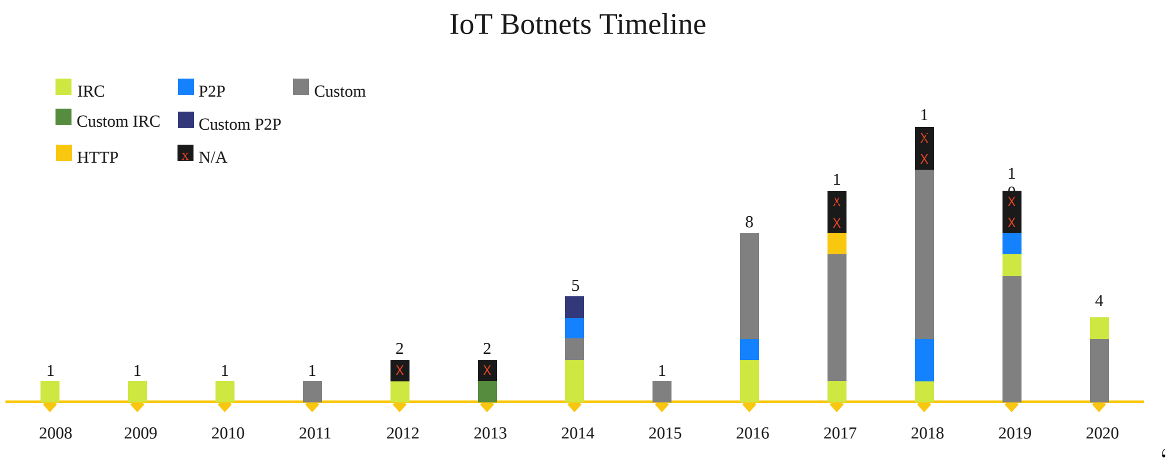

Figure 2.8: Timeline of IoT botnets per protocol [45]

Chapter 3

Related work

This chapter provides a brief explanation of botnet detection techniques and a summary of the related work done in the field of botnet detection. Botnet detection and botnet analysis have gained a big interest in recent years, and different solutions have been proposed in academia. Botnet detection techniques are classified as either honeynet-based or IDS-based. Honeynet-based botnet detection is based on setting up honeynets - a network set up with intentional vulnerabilities. However, honeynets are mostly useful for understanding botnet technology and characteristics but do not necessarily detect bot infection.

The second category of botnet detection is based on Intrusion Detection Systems (IDS). An IDS observes and identifies the presence of an active attack by inspecting network traffic. IDSs are further divided into two subcategories: signature-based and anomaly-based.

3.1 Honeynet-based Botnet Detection

Honeynet-based botnet detection became a popular way to improve network security. A honeynet is a computer network composed of simulated devices used to improve the security of networks and systems. In terms of botnet detection, the primary purpose of honeynets are to collect and understand the characteristics of botnets. From a honeynets' collected data, it is possible to perform an analysis of a botnet to discover C&C servers, unknown vulnerabilities, techniques used by the attacker, and the attacker's motivation [43].

As honeynets became popular in monitoring and defense systems, attackers began to look for methods to avoid honeynets. Zou et al. introduced various means by which attackers could detect honeypots in their constructed botnets. Honeypot detection is based on the assumption that cybersecurity engineers have liability limitations, and their deployed honeypots are not allowed to participate in real attacks [46]. Based on this assumption, honeypots can be detected by attackers in two ways. First, attackers can check whether the compromised machines in the botnet can successfully send out unmodified malicious traffic to the attackers' sensors. Second, attackers can check whether the bot controller in their botnet can successfully relay potential attack commands. Finally, the authors presented a "two-stage reconnaissance" worm. The worm is designed to automatically construct a structured peer-to-peer botnet and detect and remove infected honeypots.

Honeynets are essential for understanding the botnets' characteristics, and they are easy to build and deploy with minimum cost and resource requirements. However, they have several limitations, which are mainly associated with the size of the honeynet. Since honeynets are able to report information only about the infected machines placed as traps, the drawbacks are limited scalability and limited interaction with malicious activities.

3.2 IDS-based Botnet Detection

An Intrusion Detection System (IDS) observes and identifies the presence of an active attack by inspecting the network traffic for known attack vectors. IDSs are classified as either signature-based IDS or anomaly-based IDS [43]. Signature-based techniques in botnet detection use signature matching to detect known bots, while anomaly-based techniques consider several different network traffic anomalies.

3.2.1 Signature-based Techniques

Signature-based techniques detect only known botnets through signature matching using an IDS, such as Snort [5]. The basic idea is to extract unique patterns of information from packets of monitored traffic, mark such patterns, and register them in a knowledge database of malicious traffic. From the signature database, it is not difficult to perform the necessary detection tasks, simply by comparing every byte in the packet.

Ramachandran et al. developed a signature-based technique of botnet detection that uses a DNS Blackhole List (DNSBL) to blacklist botnet related domains [41]. This method uses techniques and heuristics for detecting DNSBL reconnaissance activity. However a downside to this is that botmasters can perform lookups against the DNSBL in order to determine whether bots have been blacklisted.

3.2.2 Anomaly-Based Techniques

Anomaly-based botnet detection examines network traffic anomalies that may indicate the presence of malicious bots in the network. The anomalies are high network latency, high traffic volume, traffic on unusual ports, and unusual system behavior [42]. Anomaly-based techniques are further divided into host-based and network-based.

3.2.2.1 Host-based techniques

In a host-based approach, the individual machine (host) is monitored to find any suspicious behavior, including its processing overhead, and access to suspicious files.

Masud et al. proposed a multiple log-file based temporal correlation technique for detecting C&C traffic [27]. By temporarily correlating two host-based log files, the authors can detect bot activity in a host machine. Relevant features were extracted from host-based log files to detect C&C traffic, and experimental results showed better overall performance compared to the other published techniques.

Liu et al. designed BotTracer to detect bot's evidences using the virtual machine techniques. The evidences of bots that authors try to detect are:

1. The startup of a bot is automatic.
2. A bot establishes a command and control channel with the botmaster.
3. A bot will perform local or remote attacks.

The results showed that BotTracer successfully detected all the bots in the performed experiments without any false negatives. One of BotTracer's biggest limitations however is that it relies on multiple assumptions. One of these assumptions is that the bot can't detect virtual machines, although the authors admit that in practice there are many techniques for virtual machine detection.

Despite its importance, the host-based approach is usually complex and difficult to scale because a monitoring tool needs to be installed on each machine in the network in order for the method to be functional.

3.2.2.2 Network-based techniques

Network-based botnet detection techniques are the most commonly used techniques today [12]. In a network-based approach, botnets are detected either passively or actively. Active monitoring injects packets into the network to measure network response and passive monitoring techniques observe data traffic in the network and look for suspicious communications that may be provided by bots or C&C servers.

Gu et al. implemented an active monitoring prototype system called BotProbe, based on an active botnet probing technique [22]. This active botnet probing technique aims to separate botnet C&C conversations from human conversation. The results on multiple real-world IRC bots demonstrated that BotProbe successfully identified botnet conversations with a low response time. The real world deployment showed that the technique has a low false-positive rate (FPR) on human conversations. However, the deployment of BotProbe is limited to a botnets that use C&C servers in the form of chat.

On the other hand, passive monitoring usually has slower detection times than active monitoring techniques. This is often caused by observation of the following phases: multiple different infection stages (BotHunter) [19], multiple instances of communications/activities (Botsniffer) [21], and a long communication/activity time (BotMiner) [20].

In the network traffic, common patterns are associated with each phase of a botnet's life cycle. Such similarities, in both WAN and LAN traffic, have been studied by several authors to perform botnet detection [20, 15]. However, most detection techniques focus on the specific network protocol used by a particular group of botnets.

3.3 Detection Techniques of IRC Botnets

Since this work is focused on the detection of botnets that use the IRC protocol, the following subchapter discusses the general overview of techniques used for the detection of IRC botnets.

One of the first studies to detect IRC botnets was by Cooke et al. in 2005. The authors studied the effectiveness of detecting botnets by directly monitoring IRC communication and showed that a more sophisticated approach was required [16]. To identify suspicious activity, the authors proposed an approach to detect botnets by ensembling secondary detection information from existing proven detection systems. However, no specific results of the described approach were provided.

Binkley et al. presented the first anomaly-based system for detecting IRC-based botnet meshes [13]. TCP-based anomaly detection, IRC tokenization, and IRC message statistics, were combined to create a system that can detect botnets. The authors also elaborated on how statistical measures, such as JOIN, PING, PONG, and PRIVMSG counts, can quickly reveal bot servers. However, as in the previous case, no specific results of the described approach were provided.

Rishi is an example of a signature-based tool for detecting IRC botnets [18]. Rishi tracks IRC bots using well-known IRC bot nickname patterns as signatures. Rishi observes protocol messages and uses n-gram analysis together with a scoring function and blacklists/whitelists to detect IRC characteristics. Upon detection, Rishi determines the IP address of the C&C server, channels the bot joined, and the additional parameters which were set. However, Rishi is not a fully automated tool because the botnet is not completely detected without a large number of false positives. Therefore, Rishi serves more as an extension to already existing IDS systems for providing additional information rather than a standalone tool. Another limitation of Rishi is that because Rishi depends on regular expressions as signatures to identify infected bot machines automatically, the tool is limited to only detect bots for which an expression exists.

Strayer et al. proposed a network-based approach where the authors search for evidence of botnet activity by examining flow characteristics such as bandwidth, duration, and packet timing [44]. The authors created a method that eliminated traffic that is unlikely to be a part of a botnet. Then, non-eliminated traffic was classified as a group that is likely to be part of a botnet. Finally, the remaining traffic was analyzed to detect botnet communications patterns. The most powerful method correctly identified between 88% and 99% of the IRC flows and non-IRC flows.

Claudio Mazzariello introduced a network traffic analysis architecture and described a behavioral model capable of identifying botnet-related activities based on IRC user behavior in a channel. Using methods such as SVM or J48, an accuracy of 100% was achieved, but as the author mentioned, further investigation is needed in order to understand whether the obtained results were biased or not [28].

Livadas et al. focused on identifying command and control traffic of IRC-based botnets using machine learning [24]. The identification of IRC-based botnet flows was proposed to proceed in two stages. In Stage I, authors classified flows into IRC and non-IRC flows. In Stage II, flows identified as IRC were distinguished into benign or botnet groups. The results in Stage I showed that the Naive Bayes classifier performed best in classifying IRC flows, achieving both a low false-negative score of 2.49% and a low false-positive score of 15.04%. In Stage II, no classifier was proposed because the challenging problem of accurately labeling IRC traffic as botnet and non-botnet was not solved.

Chapter 4

Datasets of IRC IoT Traffic

This chapter describes the process of collecting source data for our machine learning methods. For the experiments, we needed to collect real network traffic with malicious and benign IRC communication. The features were extracted from the collected IRC communication and used to train and evaluate our proposed detection models. For malicious IRC communication, a group of researchers in the Aposemat project infected IoT devices and captured the network traffic. For the benign IRC communication, network traffic captures of the IRC university network are used.

4.1 Malware Capture Methodology

Capturing malware is a challenging task. This section briefly describes all the steps that were followed during the execution and malware capture process.

1. **Downloading a Binary of a Malware Sample**
   To collect malware traffic, a malware sample needs to be executed to infect the targeted device. A malware sample needs to match several requirements:

   (a) Malware sample must be active.
   (b) Malware sample must be designed for the target.

   In the context of an IRC botnet, an active malware sample means that the C&C server must be alive to disseminate commands to bots. For obtaining a malware sample, we used VirusTotal [7], a website that aggregates many antivirus products and online scan engines. The second requirement is that the malware sample must be designed for the same device architecture as the device we want to infect. In our case, all RaspberryPi devices have ARM6 and ARM7 architecture.

2. **Traffic Capture Execution**
   The traffic capture is started by the capturing device using a tcpdump command:

```
#!/bin/bash
tcpdump -n -s0 -i enp5s4 -v -w
%Y-%m-%d-%H-%M-%S-192.168.1.X.pcap  host 192.168.1.X
-G 86400
```

This command captures network traffic of the infected device and stores it in packet capture (pcap) files where one file is created per one day of traffic capture. To distinguish between the multiple files that were created, filenames consist of the starting time of the capture, and the IP address of the infected device.

3. **Malware Sample Execution**
   When the traffic is being captured, the malware sample is executed in the targeted device. While executing the malware sample, the "liveliness" of the malware sample needs to be checked to see if network traffic is generated or not. When the device is infected, it should communicate with a C&C server and hence produce network traffic. If no traffic is generated by an infected device, it means that the C&C server is probably not active, and thus, the capture should be terminated.

4. **Malware Traffic Analysis**
   Once the malware sample is deemed inactive or enough amount of network traffic is captured, the execution is terminated, which consists of stopping the malware sample and the traffic capture. After that, we analyze the traffic using automatic analysis tools. Such tools consist of Zeek [9], Suricata [6], or Wireshark [8]. Each tool generates analysis files that help us better understand the content of the captured traffic.

4.1.1 Architecture of the Malware Execution Laboratory

The Aposemat IoT Laboratory was used to execute the malware and to capture the malware traffic [11]. The Aposemat IoT laboratory consists of a separate network of IoT devices which are used for different purposes. One of the purposes is to enable the infection of IoT devices and capture their network traffic. The simplified infrastructure of the laboratory is depicted in Figure 4.1. At the time of data collection, there were eight Raspberry Pi devices used for malware infection. The infected devices were connected to a switch that sends a copy of all input and output network traffic to the device, where the traffic is captured. This method is called port mirroring. The switch is connected to a router, which is then connected to the Internet.

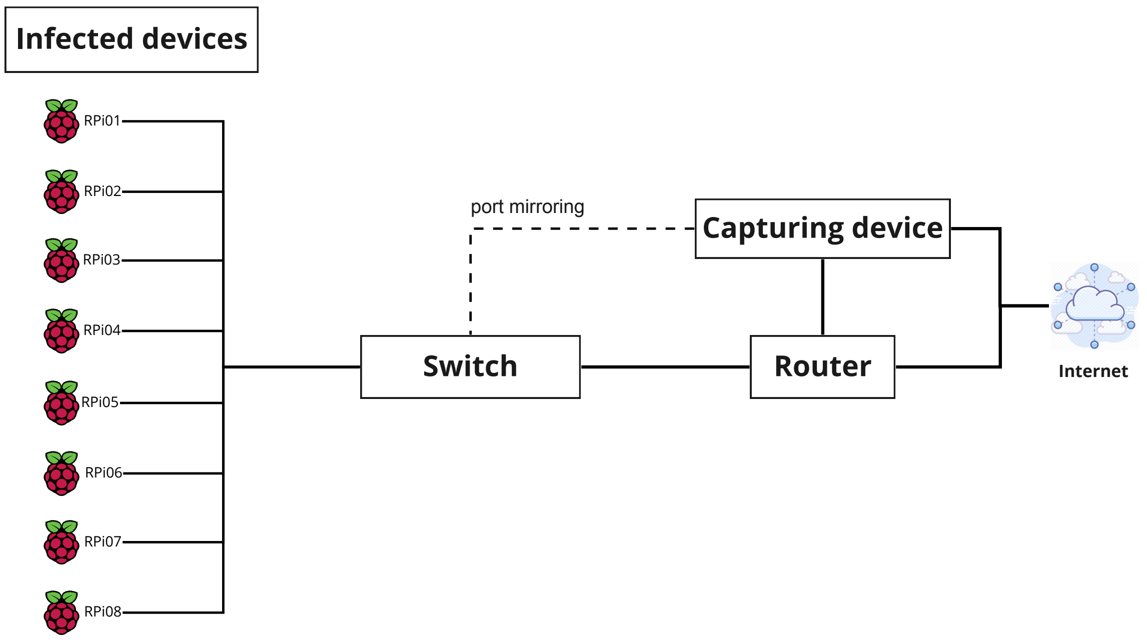

Figure 4.1: Simplified infrastructure of the Aposemat IoT Laboratory

4.2 Data Description

For the benign IRC communication, network traffic captures of the CTU IRC server were used. We performed a manual review of the IRC communication to eliminate malicious traffic labeled as benign. The manual review was performed in such a way that we went through each IRC channel in the benign traffic and checked whether the messages in that channel showed any signs of communication with a command and control server. If an IRC channel showed any sign of a command and control communication, we eliminated it from the benign traffic. Only then we could use this network traffic as a source of the benign IRC 3-tuple sessions in our research.

For the malicious IRC communication, we infected IoT devices with the following malware families: Mirai, Torii, and Shellbot. Then we monitored the network traffic of the infected devices for a certain time period in order to collect sufficiently large network captures. We used the network captures of infected devices as a source of malicious IRC communication. The network captures of infected devices are part of a recently released IoT23-dataset [10]. IoT-23 is a dataset of network traffic from IoT devices consisting of twenty malware captures executed, and three captures for benign traffic.

The URL of network traffic capture, the size of the collected traffic, and the number of infected users for each malicious network traffic capture are shown in Table 4.1.

Table 4.1: Description of the network traffic that has been used in our experiments.

Scenario	Malware family	PCAP size	Infected users
CTU-IoT-Malware-Capture-34	Mirai	11M	352
CTU-IoT-Malware-Capture-42	Torii	1.7M	250
CTU-IoT-Malware-Capture-51	Torii	2.4M	8,883
CTU-IoT-Malware-Capture-56	Shellbot	13M	652
CTU-IoT-Malware-Capture-62	Shellbot	2.8M	82
University IRC Traffic	-	2.8G	-

Once the IRC communication was obtained, there was a process to extract the data features. The features were computed using the Zeek tool, which outputed the features into a JSON logfile. Then, the logfile of each network capture was parsed, and the data features were extracted into a CSV file. Finally, the CSV file was used as an input file for the machine learning methods.

4.3 Data Visualization

To reflect the difference between the benign IRC communication and malicious IRC communication, the graphs of IRC messages are depicted in Figure 4.2 and Figure 4.3. The red dots represent channels, gray dots represent users, and blue edges represent IRC messages. It is apparent that the graph of benign traffic is denser than the graph of malicious traffic. Higher graph density is due to the larger amount of benign IRC traffic and different communication behaviors. Benign IRC users communicate in more IRC channels than malicious IRC users. On the contrary, malicious IRC users communicate mostly through a single channel, which is intended for the C&C. Another thing we can see in Figure 4.4 is that when the graph of messages in benign traffic is zoomed, it is visible that user communities are created around particular channels.

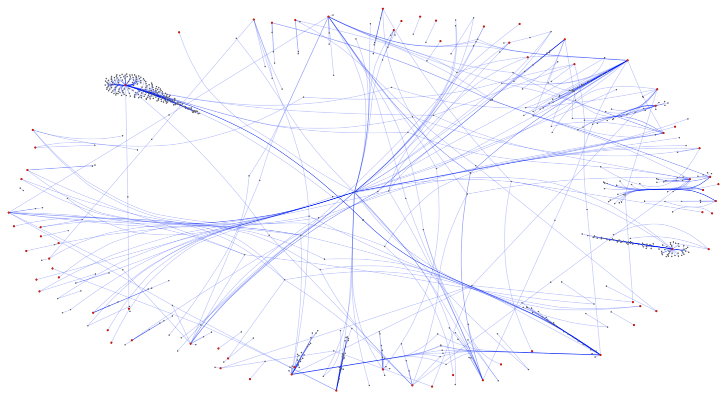

Figure 4.2: Visualization of messages in collected benign network traffic. Red dots represent channels, gray dots represent users, and blue edges represent IRC messages.

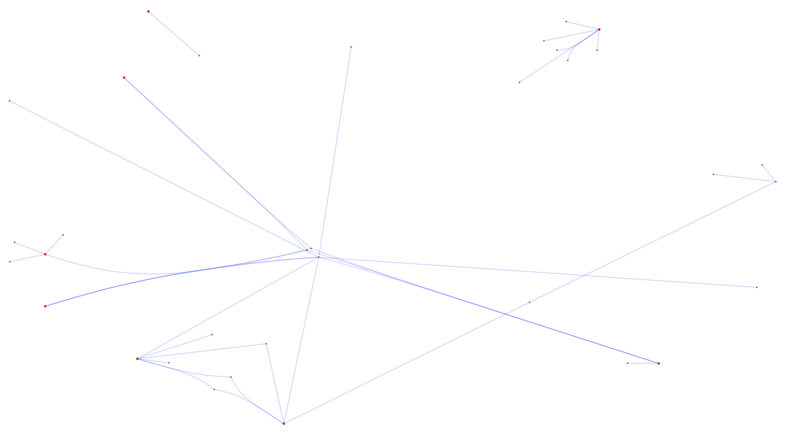

Figure 4.3: Visualization of messages in collected malicious network traffic. Red dots represent channels, gray dots represent users, and blue edges represent IRC messages.

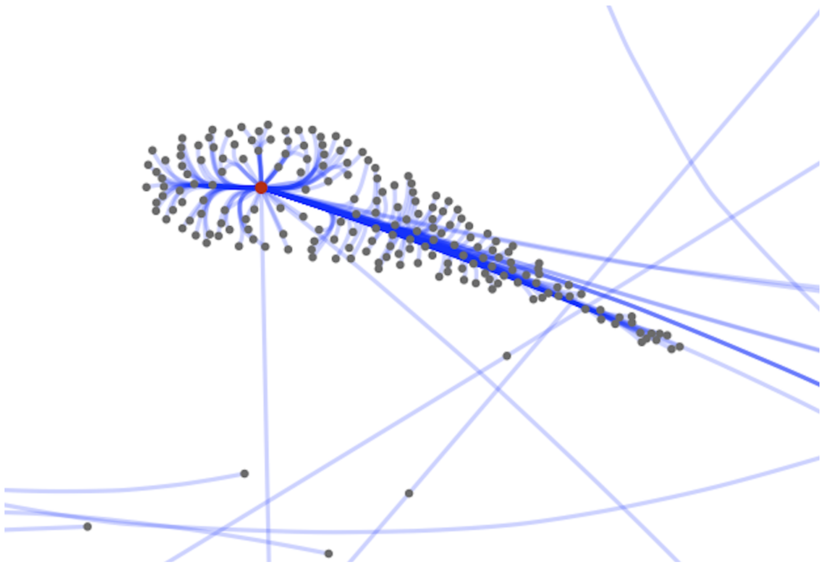

Figure 4.4: Detailed visualization of created user community around a particular channel. Red dots represent channels, gray dots represent users, and blue edges represent IRC messages.

Chapter 5

IRC 3-Tuple Session Detection

In this chapter, we propose a new technique for detecting malicious IRC sessions in a network. The whole traffic dataset is separated into IRC 3-tuple sessions. An IRC 3-tuple session is defined as a group of flows between a source IP address, a destination IP address, and a destination port.

5.1 IRC 3-Tuple Sessions Dataset

For the detection of malicious IRC 3-tuple sessions in a network, the IRC 3-Tuple Sessions dataset of network traffic was created. The dataset consists of the network traffic from the infected devices described in Chapter 4. The size of the collected traffic is shown in Table 5.1. Table 5.2 lists the datasets used in our experiments. Not all traffic captures mentioned in Chapter 4 were used, because some malicious network traffic was captured later for another part of this thesis.

Table 5.1: Description of the network traffic used in IRC 3-tuple session detection.

Scenario	Malware family	IRC 3-tuple sessions
CTU-IoT-Malware-34	Mirai	53
CTU-IoT-Malware-42	Torii	1
CTU-IoT-Malware-51	Torii	2
University IRC Traffic	-	1,470

Table 5.2: Description of the IRC 3-Tuple Sessions Dataset.

Type of IRC	IRC 3-tuple sessions	IRC messages
Malicious	56	1,255
Benign	1,470	99,250

5.2 Feature Selection

Once the data was gathered, there was a process to extract the features. In this section, we describe the complete list of extracted features for each IRC 3-tuple session. The features were chosen to give us a meaningful representation of each IRC 3-tuple session biased towards separating malicious IRC 3 from benign IRC.

To automate the extraction process, we created a Zeek package that automatically recognizes an IRC 3-tuple session, computes the features, and extracts them [39]. Zeek is an open-source network security platform that supports a wide range of traffic analysis tasks even outside of the security domain [9].

The features we extracted for each IRC 3-tuple session are:

- **Total Size** - Size of IRC messages in bytes that were sent and received in the IRC 3-tuple session.
- **Duration** - Duration of the IRC 3-tuple session in milliseconds (i.e., the difference between the time of the last message and the time of the first message in the IRC 3-tuple session).
- **Number of Messages** - The total number of messages in the IRC 3-tuple session.
- **Number of IRC Flows in the IRC 3-tuple session** - As it was mentioned before, the source port is neglected to unify communication into IRC 3-tuple sessions since the source port is randomly chosen when a TCP connection is established. We assume that bots use a higher number of source ports than real users since the number of IRC flows of bots was higher than the number of IRC flows of the real users.
- **Message Periodicity** - Bots that are controlled by a botmaster use IRC for sending commands periodically, so we wanted to obtain the value of periodicity [21]. To do that, we created a method that would return a number between 0 and 1, one if the message sequence is perfectly periodic, zero if the message sequence is not periodic.
- **Message Word Entropy** - To consider whether the user sends the same message multiple times in a row, or whether the message contains a limited number of words, we compute a word entropy across all of the messages in the IRC 3-tuple session.
- **Mean of Username Special Characters** - An average usage of non-alphabetic characters in the username is computed. This feature identifies whether the username of the user in the IRC 3-tuple sessions is randomly generated or not.
- **Mean of Message Special Characters** - If the messages contain many sent commands, it will most likely contain a lot of different characters than the ordinary messages. The majority of commands consist of IP addresses, device directories, or URLs. With this feature, we obtain the mean usage of non-alphabetic characters across all messages in the IRC 3-tuple session.

The next subsection is focused on describing the computation process of the complex features in detail, specifically message periodicity, message word entropy, mean of username special characters, and mean of message special characters.

5.2.1 Message Periodicity

There are four stages in the creation of the message periodicity feature: computing time differences between messages, applying fast Fourier transform, splitting the messages by the most significant period, and computing the Normalised Mean Squared Error. The computation process is illustrated in Figure 5.1.

1. **Computing time differences between messages**
   For each message, a set of attributes is stored where one of them is the time the message was sent. We use that attribute to compute time differences between consecutive pairs of chronologically-sorted messages.

2. **Applying fast Fourier transform**
   On the sequence of time differences between messages, we apply a fast Fourier transform (FFT). The fast Fourier transform is an effective algorithm for computing the discrete Fourier transform, which is a function to express time sequence as the sum of periodic components and to recover a signal from those components [30]. The output of the FFT is a sequence of numbers. The larger the number on the given position of the output, the larger the amplitude on the given position. Thus it has a more significant influence on the periodicity of the data.

3. **Splitting messages by most significant frequency**
   The position of the largest element in the FFT's output represents the frequency, which is the most significant from all other frequencies. To compute the quality of the most significant frequency, we split the data by that frequency.

4. **Computing Normalised Mean Squared Error (NMSE)**
   The Normalized Mean Squared Error is computed from split messages and returns the resulting number in the interval between 0 and 1, where 1 represents the perfectly periodic messages, and 0 represents not periodic messages at all. The Normalized Mean Squared Error is computed as follows:

$$NMSE(td_p) = \frac{MSE(td_p, \overline{td_p})}{MSE(td_p, 0)} = \frac{||td_p - \overline{td_p}||_2}{||td_p||_2} \tag{5.1}$$

   where td_p represents a list of time differences between messages in the p-th period and $\overline{td_p}$ represents the mean of the time differences between messages in the p-th period.

5.2.2 Message Word Entropy

To analyze whether the user sends the same message multiple times in a row, or whether the message contains a limited number of different words, we compute the word entropy across all the messages in the IRC 3-tuple session. By the term word entropy, we mean a Shannon entropy of words. The word entropy is computed as follows:

$$H(X) = -\sum_{i=1}^{n} p_i \log(p_i) \tag{5.2}$$

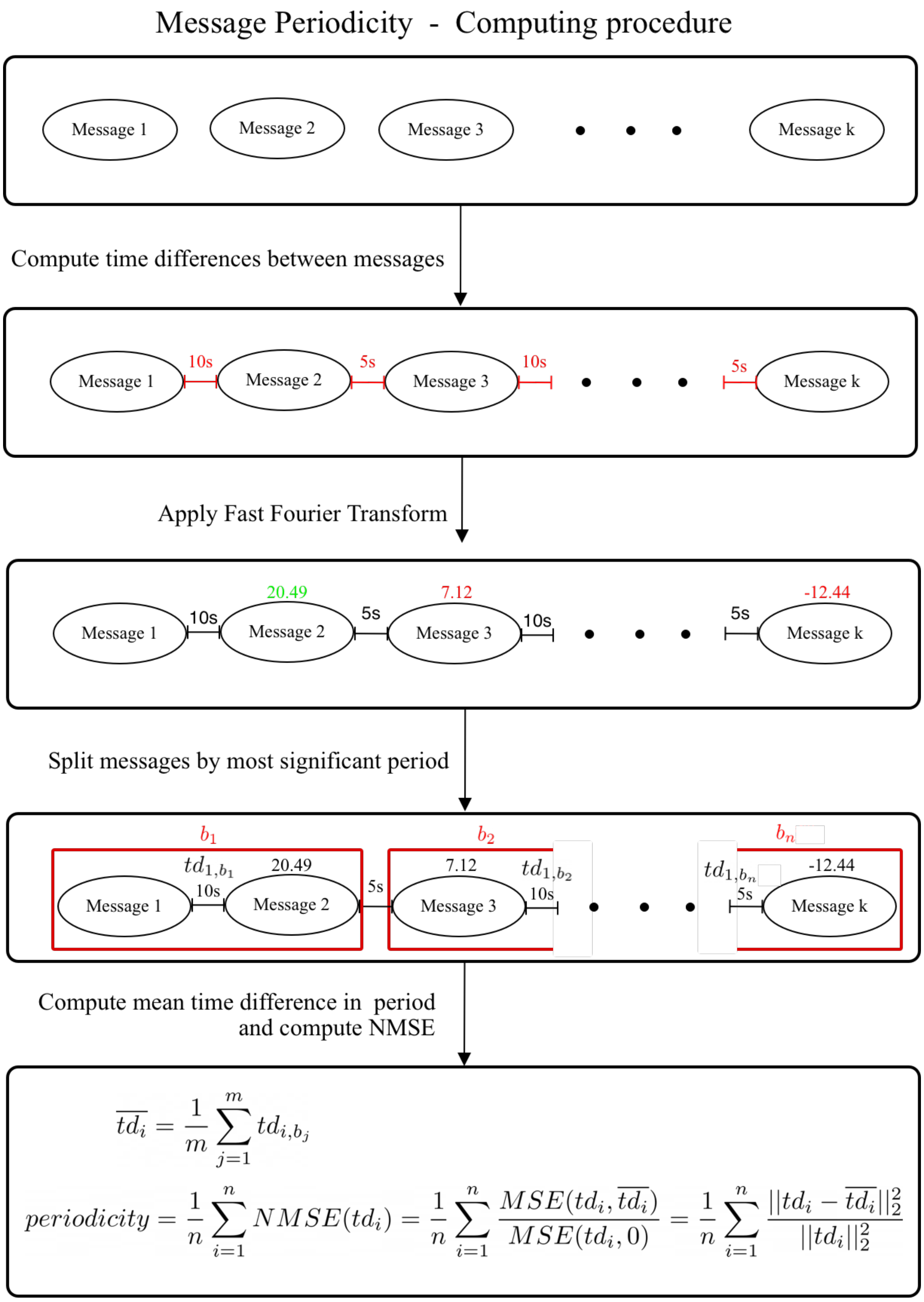
Message Periodicity - Computing procedure
Message 1
Message 2
Message 3
Message k
Compute time differences between messages
10s
5s
10s
5s
Apply Fast Fourier Transform
20.49
7.12
-12.44
Split messages by most significant period
b_1
b_2
b_n
td_{1,b_1}
td_{1,b_2}
td_{1,b_n}
Compute mean time difference in period and compute NMSE
$\overline{td_i} = \frac{1}{m}\sum_{j=1}^{m} td_{i,b_j}$
$periodicity = \frac{1}{n}\sum_{i=1}^{n} NMSE(td_i) = \frac{1}{n}\sum_{i=1}^{n}\frac{MSE(td_i,\overline{td_i})}{MSE(td_i,0)} = \frac{1}{n}\sum_{i=1}^{n}\frac{||td_i - \overline{td_i}||_2^2}{||td_i||_2^2}$

Figure 5.1: Illustration of computing message periodicity feature. There are four stages in the creation of the message periodicity feature: computing time differences between messages, applying fast Fourier transform, splitting the messages by the most significant period, and computing the Normalised Mean Squared Error.

where X represents a discrete set of words, n represents the number of words, and p_i represents the frequentist probability of the word in the IRC 3-tuple session, computed as follows:

$$p_i = \frac{\text{Number of occurrences of the word in connection}}{\text{Total amount of words in connection}}$$

5.2.3 Mean of Username Special Characters

This feature is extracted to identify whether the username of the user in the IRC 3-tuple session was randomly generated or not. We compute the average usage of non-alphabetic characters in the username. We matched non-alphabetic characters with the following regex:

$$r'[a-zA-Z] \tag{5.3}$$

The regex matches all characters that are neither lowercase alphabetic characters nor uppercase alphabetic characters.

5.2.4 Mean of Message Special Characters

If the message contains many sent commands, it will most likely contain a lot of different characters than the ordinary message. With this feature, we obtain the mean usage of non-alphabetic characters across all messages in the IRC 3-tuple session. We apply the same procedure of matching special characters for each message as in the previous case: we match non-alphabetic characters by regex, and then we divide the number of matched characters by the total number of message characters. Finally, we compute an average of all the obtained values for each message.

5.2.5 Features Visualization

By placing data into a visual context, we projected the feature vectors into lower dimensions (this process is also known as a low-dimensional embedding). We used different types of dimensional reduction methods to project the feature vectors, linear and nonlinear. Firstly, we applied the Principal Component Analysis (PCA) 5.2. Then we projected the data using the t-SNE 5.3 and the Uniform Manifold Approximation (UMAP) 5.4 algorithms. Visualization helped us explore selected features as a suitable representation of the model since malicious IRC 3-tuple sessions are projected close to each other.

5.3 Detection using Machine Learning

To detect malicious IRC 3-tuple sessions, we decided to use two types of machine learning methods: unsupervised and supervised. Unsupervised methods look for previously undetected patterns in the data. On the contrary, supervised methods make use of labeled input data and try to predict the output.

For the unsupervised methods, we used K-Means and Hierarchical Clustering. For the supervised methods, the following methods were used: Linear Classifier, Logistic Regression, SVM, Random Forest, and XGBoost. The source code of all machine learning models is published in the Stratosphere's git repository [38]. The following subsection describes the training and evaluation process used to fit these models.

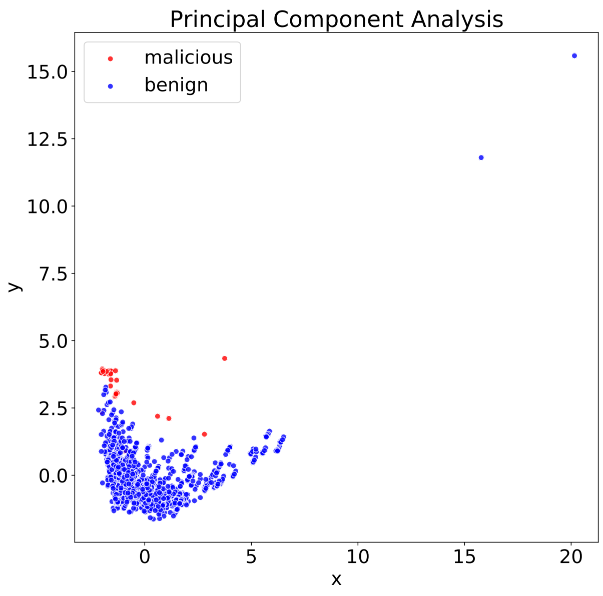

Figure 5.2: PCA visualization of IRC 3-tuple sessions in the IRC 3-Tuple Sessions dataset. Red dots represent malicious IRC 3-tuple sessions, and blue dots represent benign IRC 3-tuple sessions.

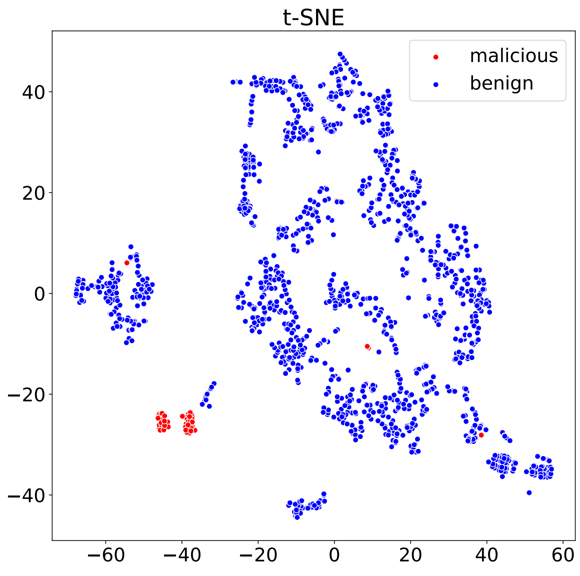

Figure 5.3: t-SNE visualization of IRC 3-tuple sessions in the IRC 3-Tuple Sessions dataset. Red dots represent malicious IRC 3-tuple sessions, and blue dots represent benign IRC 3-tuple sessions.

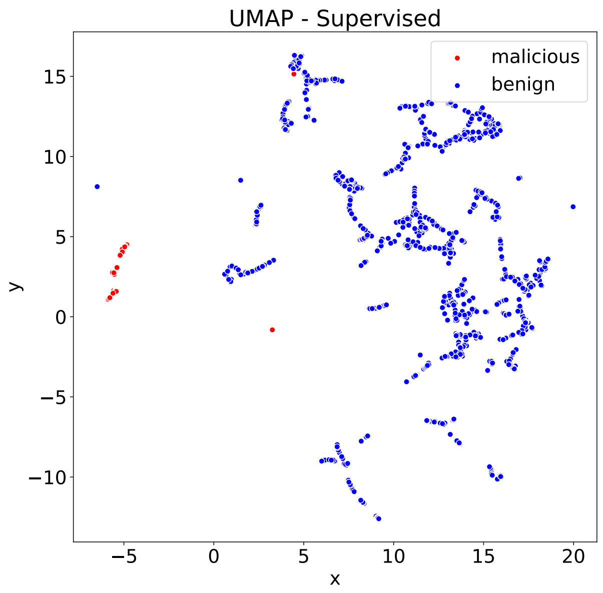

Figure 5.4: UMAP visualization of IRC 3-tuple sessions in the IRC 3-Tuple Sessions dataset. Red dots represent malicious IRC 3-tuple sessions, and blue dots represent benign IRC 3-tuple sessions.

5.3.1 Evaluation Process

As the number of non-malicious samples and malicious samples differ, the accuracy metric would not indicate the quality of the machine learning model well. As a metric of a given machine learning model's quality, we chose to use the F1-score. F1-Score is the harmonic mean of precision and recall, and it is a good fit for our data models.

In order to create the most accurate machine learning model possible we tuned the hyper-parameters of the models. Hyper-parameters are parameters that affect the learning process of a machine learning model. To determine the optimal hyper-parameter values for a given model, grid-search cross-validation was applied. Hyper-parameter tuning was performed in order to create the most accurate machine learning model possible.. To diagnose bias and variance in trained models, we generated learning curves for each of them.

In order to train the supervised machine learning model, the dataset was divided into a training set and a test set. To have a sufficient amount of malicious samples in the test set, it consists of 40% of samples, and the training set consists of the remaining 60% of samples, while the original proportion of both target classes was preserved in both sets. On the contrary, for all unsupervised methods, the dataset was not split, and we use the full amount of samples in the training phase. To evaluate the resulting model, we compared the labels with the predicted classes and computed the resulting score.

After the evaluation of all machine learning models on the imbalanced dataset, class balancing was performed. Most machine learning classification algorithms are sensitive to an imbalanced dataset. An imbalanced dataset refers to a dataset where we have an highly unequal number of instances in the predictor classes. As a result, an imbalanced dataset will bias the prediction model towards the majority class. In the case of the IRC 3-Tuple Sessions dataset, where the majority of dataset instances are benign, a machine learning model that has been trained and evaluated on that dataset could now predict "benign" for all samples and still have very high accuracy. To avoid the problem of biasing a prediction model to the majority class, there are several methods that can be used. To balance classes in the dataset, we used two techniques: undersampling and oversampling. By undersampling a dataset, a random subset of samples is selected from the majority class to match the number of samples in the minority class. Using oversampling, a random subset of samples is duplicated from the minority class to match the number of samples in the majority class. The visualization for undersampling and oversampling is depicted in Figure 5.5.

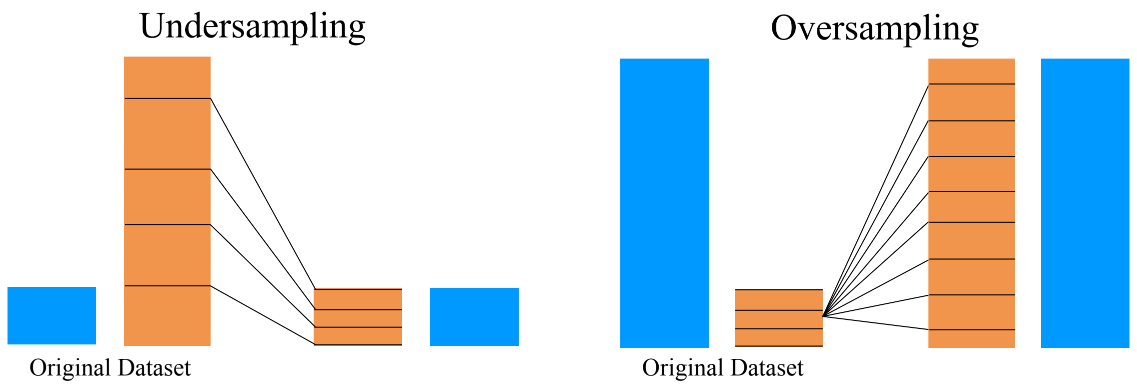

Figure 5.5: Description of class balancing methods - undersampling and oversampling.

5.3.2 Results

Results in Table 5.3 show that unsupervised methods do not have a good performance for our problem, and the highest achieved F1-score is only 10.02%. Among supervised methods, The worst performance was for the Linear Classifier that achieved the F1-score of 87.50%. Logistic Regression, together with SVM, achieved the F1-score of 93.33%. All methods were outperformed by Random Forest and XGBoost, which achieved an F1-score of 96.24%.

The learning curve of Random Forest is visualized in Figure 5.6. The figure shows that with increasing the number of samples, the model cannot fit perfectly the training set anymore, so the training error becomes larger. However, as the number of samples gets larger, the model starts performing well on both sets of samples, and the training error and the test error become small. Learning curves for the rest of the evaluated models are attached in Appendix A.

The results showed that the linear models could not capture the complexity of the data that we have as well as ensemble methods. We knew that ensemble methods are more powerful methods in general, and the results confirmed it.

Table 5.3: IRC 3-Tuple Session Detection: Score of machine learning methods trained on the imbalanced IRC 3-Tuple Sessions dataset.

Method	Hyper-parameters	TN	FN	TP	FP	Accuracy	F1-score	FPR
K-Means	k: 3	887	558	4	52	59.36%	1.29%	5.54%
Hierarchical Clustering	-	515	930	52	4	37.77%	10.02%	0.77%
Linear Classifier	-	574	5	21	1	99.00%	87.50%	0.17%
Logistic Regression	-	577	2	21	1	99.50%	93.33%	0.17%
SVM	kernel: RBF C : 0.01 γ : 0.1	577	2	21	1	99.50%	93.33%	0.17%
Random Forest	estimators: 10 tree max depth: 4	579	0	20	2	99.67%	95.24%	0.34%
XGBoost	estimators: 5 tree max depth: 2 γ: 0 α: 0.01 λ: 0.01	579	0	20	2	99.67%	95.24%	0.34%

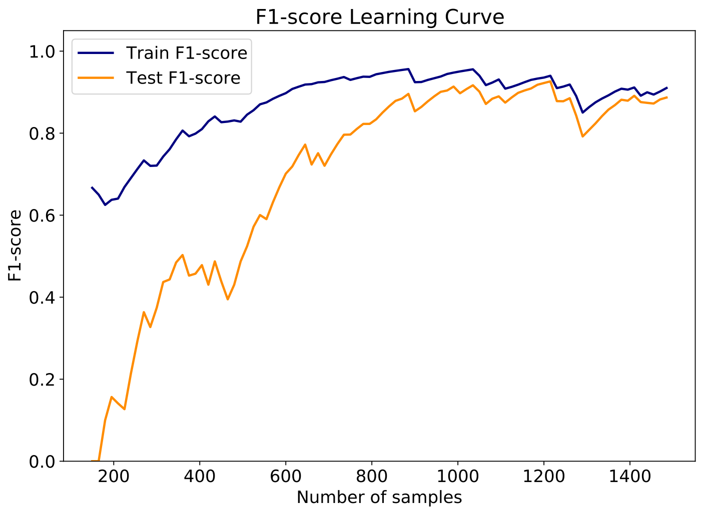

Figure 5.6: Learning curve of Random Forest Classifier

5.3.3 Feature Importance

We interpreted evaluated models to see what are the most important features. Figure 5.7 visualizes the most important features of the Random Forest classifier. The most important feature of the final model is the mean of message special characters. The final decision is made together with another five features: message periodicity, total packet size, message word entropy, mean of username special characters, and duration. On the contrary, the message count, and the number of flows in the IRC 3-tuple sessions did not have big importance for the final decision. Since IRC usually has many messages exchanged, the message count is not as important as the content of the messages.

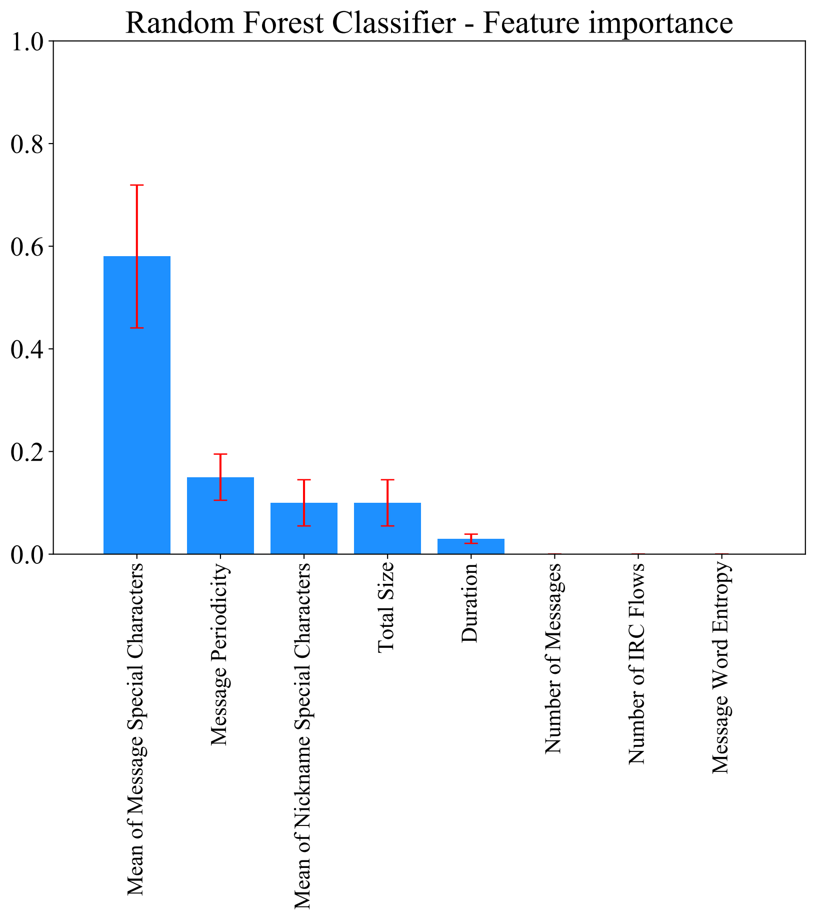

Figure 5.7: Feature importance of the best performed model - Random Forest Classifier.

5.3.4 Class Balancing

To balance classes in the IRC 3-Tuple Sessions dataset, we used undersampling and oversampling. By undersampling the IRC 3-Tuple Sessions dataset, a random subset of malicious samples was selected to match the number of benign samples. On the contrary, by oversampling the IRC 3-Tuple Sessions dataset, a random subset of malicious samples is duplicated to match the number of samples in the benign class. Description of the undersampled IRC 3-Tuple Sessions dataset and oversampled IRC 3-Tuple Sessions dataset is shown in Table 5.4 and Table 5.5, respectively.

We trained detection models on the balanced IRC 3-Tuple Sessions datasets to verify the correctness of our results achieved on the original IRC 3-tuple sessions dataset. The results for the undersampled dataset are shown in Table 5.6. We can notice that unsupervised methods performed better on the undersampled IRC 3-Tuple Sessions dataset. K-Means achieved an F1-score of 4.65% and Hierarchical Clustering improved by more than 80% and achieved

Table 5.4: Description of undersampled IRC 3-Tuple Sessions dataset.

Type of IRC	IRC 3-tuples	IRC messages
Malicious	56	1,255
Benign	56	3,780

Table 5.5: Description of oversampled IRC 3-Tuple Sessions dataset.

Type of IRC	IRC 3-Tuple Sessions	IRC messages
Malicious	1,470	33,258
Benign	1,470	99,250

an F1-score of 94.34%. For the supervised methods, not all supervised methods performed better. The results showed that the score improved for Linear Classifier by achieving an F1-score of 91.89%. Logistic Regression, together with SVM, outperformed the rest of the methods by achieving an F1-score of 97.14%. Random Forest achieved an F1-score of 96.97%. On the contrary, the F1-score of XGBoost worsened by achieving 94.12%.

The results for the oversampled IRC 3-Tuple Sessions dataset are shown in Table 5.7. It is apparent from the table that the results improved for the majority of models. K-Means improved by more than 70% and achieved an F1-score of 77.70%. Hierarchical Clustering achieved an F1-score of 97.03%. Linear Classifier did not perform well and achieved only an F1-score of 87.50%. Logistic Regression achieved an F1-score of 0%. Random Forest Classifier performed the best among others and achieved an F1-score of 99.88%. XGBoost achieved an F1-score of 99.88%.

By applying the class balancing methods, the results have improved for most models. Compared to undersampling, oversampling worked better in most cases which is caused by having a small amount of samples in a minority class of the original dataset. However, the quality of the best performing model, Random Forrest Classifier, improved in both class balancing methods.

5.4 Deployment of IRC 3-tuple Session Detection in SLIPS

We integrated a new module that detects malicious IRC 3-tuple sessions in the network traffic into the Stratosphere Linux IPS (Slips) system [4]. Slips is a behavioral-based intrusion detection and prevention system that uses machine learning algorithms to detect malicious behaviors. By integrating multiple modules into SLIPS, SLIPS uses the outputs of the modules to create a profile for each IP address within network traffic to prevent system intrusion. SLIPS uses Zeek to analyze network traffic and it integrates the Zeek package that we created to automatically recognize, compute, and extract IRC 3-tuple sessions. Our newly created model integrates our best performed detection model that use the output features from the Zeek package as an input to the detection model.

Table 5.6: IRC 3-Tuple Session Detection: Score of machine learning methods trained on undersampled IRC 3-Tuple Sessions dataset.

Method	Hyper-parameters	TN	FN	TP	FP	Accuracy	F1-score	FPR
K-Means	k: 2	28	28	2	54	26.79%	4.65%	65.85%
Hierarchical Clustering	-	56	0	50	6	94.64%	94.34%	9.68%
Linear Classifier	-	14	3	17	0	91.18%	91.89%	0%
Logistic Regression	-	16	1	17	0	97.06%	97.14%	0%
SVM	kernel: RBF C : 1000 γ : 0.001	16	1	17	0	97.06%	97.14%	0%
Random Forest	estimators: 10 tree max depth: 1	15	1	16	0	96.88%	96.97%	0%
XGBoost	estimators: 5 tree max depth: 2 γ: 0 α: 0.01 λ: 0.01	15	2	16	0	93.94%	94.12%	0%

Table 5.7: IRC 3-Tuple Session Detection: Score of machine learning methods trained on oversampled IRC 3-Tuple Sessions dataset.

Method	Hyper-parameters	TN	FN	TP	FP	Accuracy	F1-score	FPR
K-Means	k: 3	1369	735	710	9	72.80%	77.70%	0.65%
Hierarchical Clustering	-	1370	75	1436	9	97.09%	97.03%	0.65%
Linear Classifier	-	424	9	422	12	99.00%	87.50%	2.75%
Logistic Regression	-	426	7	423	11	97.92%	97.93%	2.52%
SVM	kernel: RBF C : 0.01 γ : 0.1	433	0	431	3	99.65%	99.65%	0.69%
Random Forest	estimators: 10 tree max depth: 4	433	0	433	1	99.88%	99.88%	0.23%
XGBoost	estimators: 5 tree max depth: 2 γ: 0 α: 0.01 λ: 0.01	433	0	430	4	99.54%	99.54%	0.92%

Chapter 6

IRC Channel & User Detection

In this chapter, a new detection technique of malicious IRC channels and IRC users is proposed. In Chapter 5, the technique of detecting IRC 3-tuple sessions was proposed. However, this technique detects IRC 3-tuple sessions, and it does not specify which IRC channel has been used for C&C nor which IRC user has been detected. Detection of malicious IRC channels and detection of malicious IRC users uses the same dataset of network captures, and the evaluation process is the same, but the extracted features differ. Therefore, the evaluation results differ too.

This chapter is divided into four sections. The first section discusses the dataset that is the same for both sub-problems, the second section describes the evaluation process, the third section elaborates on the IRC channel detection, and the fourth section elaborates on the IRC user detection.

6.1 IRC Channel & User Dataset

The number of IRC channels and IRC users in the network traffic is significantly lower than the number of IRC 3-tuple sessions. Therefore, the IRC 3-tuple Sessions dataset does not have enough malicious data for creating a model for detecting malicious IRC channels and IRC users. Thus, the IRC 3-tuple Sessions dataset of network traffic has been enhanced with new malicious network traffic captures that have resulted from infecting more devices with another set of malware samples. The capture of benign network traffic is sufficiently big enough even for the second part of this thesis, and it remains unchanged. The size of the collected traffic is shown in Table 6.1. Table 6.2 lists the dataset used in our experiments.

Table 6.1: Description of the network traffic used in IRC Channel & User Detection.

Scenario	Malware family	IRC Channels	IRC Users
CTU-IoT-Malware-34	Mirai	1	6
CTU-IoT-Malware-42	Torii	1	1
CTU-IoT-Malware-51	Torii	3	3
CTU-IoT-Malware-56	Shellbot	2	6
CTU-IoT-Malware-62	Shellbot	1	3
University IRC Traffic	-	73	563

Table 6.2: Description of the IRC Channel & User dataset.

Type of IRC	IRC channels	IRC users	IRC messages
Malicious	8	19	7643
Benign	73	563	99250

6.2 Detection using Machine Learning

We used two types of machine learning methods to detect malicious IRC channels and IRC users: unsupervised and supervised. For the unsupervised methods, we used K-Means and Hierarchical Clustering For the supervised methods, the following methods were used: Linear Classifier, Logistic Regression, SVM, Random Forest, and XGBoost. The source code of all machine learning models is published in the Stratosphere's git repository [38].

6.2.1 Evaluation Process

As the numbers of non-malicious samples and malicious samples differ as in Chapter 5, we choose to use F1-score to measure the quality of the machine learning models. For all supervised methods, the hyper-parameter tuning was performed using grid search cross-validation to provide a model as accurate as possible. To diagnose bias and variance in trained models, we generated learning curves for each of them.

In order to train the supervised machine learning model, the dataset was divided into a training set and a test set. To have a sufficient amount of malicious samples in the test set, it consists of 40% of samples, and the training set consists of the remaining 60% of samples, while the original proportion of both target classes was preserved in both sets. On the contrary, for all unsupervised methods, the dataset was not split, and we use the full amount of samples in the training phase. To evaluate the resulting model, we compared the labels with the predicted classes and computed the resulting score.

After the evaluation of all machine learning models on the imbalanced dataset, class balancing was performed. Most machine learning classification algorithms are sensitive to an imbalanced dataset. An imbalanced dataset refers to a dataset having an unequal number of instances in the predictor classes. As a result, an imbalanced dataset will bias the prediction model towards the majority class. In the case of the IRC Channel & User dataset, where the majority of dataset instances are benign, a machine learning model that has been trained and evaluated on that dataset could now predict "benign" for all samples and still gain very high accuracy. To avoid the problem of biasing a prediction model to the majority class, there are several methods that can be used. One method is to balance classes in the dataset with the same amount of benign samples and malicious samples. To balance classes in the dataset, we used two techniques, which are undersampling and oversampling. By undersampling a dataset, a random subset of samples is selected from the majority class to match the number of samples in the minority class. On the contrary, using an oversampling, random subset of samples is duplicated from the minority class to match the number of samples in the majority class. The visualization of undersampling and oversampling is depicted in Figure 5.5.

6.3 IRC Channel Detection

IRC channel is a place where users meet and chat. IRC-based botnets use the IRC channel to send commands to bots to control them. This section describes a method used for the detection of IRC channels that are used by botnets for C&C. We separated the whole traffic per IRC channels. For every IRC channel, features were extracted from the network traffic and used as data input into detection models.

6.3.1 Feature Selection

In this section, we describe the complete list of features that were extracted for each IRC channel from the IRC channel & user dataset. The features were chosen to provide us with a meaningful representation of the IRC channel biased towards separating malicious IRC channels from benign IRC channels. The features that were extracted for each IRC channel are:

- **Message Periodicity** - We suppose that bots that are controlled by botnet master use IRC for sending commands periodically. To obtain the value of periodicity, we created a method that would return a number between 0 and 1 - i.e. one if the message sequence is perfectly periodical, fzero if the message sequence is not periodical at all. The computation process of the feature was described in Section 5.2.1.

- **Mean of Message Special Characters** - If the message contains many sent commands, it will most likely contain a lot of different characters than the ordinary message. With this feature, we obtain the mean usage of non-alphabetic characters across all sent messages by the IRC user. The computation process of the feature was described in Section 5.2.4.

- **Message Word Entropy** - To consider whether the IRC user sends the same message multiple times in a row, or whether the message contains a limited number of words, we compute a word entropy across all of the IRC messages sent by the IRC user. The computation process of the feature was described in Section 5.2.2.

- **Maximal Channel Users** - Number of maximal amount of users in the IRC channel that was recorded during the traffic capture.
- **Mean of Levenshtein Distance of IRC Messages** - Mean of pairwise Levenshtein distance of IRC messages in the IRC channel.
- **STD of Levenshtein Distance of IRC Messages** - Standard deviation (STD) of pairwise Levenshtein distance of IRC messages in the IRC channel.
- **Channel's Language** - Recognized language from IRC messages in the IRC channel.
- **Message Frequency** - Frequency of IRC messages in the IRC channel. Computed as a time of a network capture divided by a number of IRC messages sent in the IRC channel.
- **Number of Active Users** - The number of users in the IRC channel that sends at least one message during network traffic capture.
- **One-channel Users Ratio** - The ratio of IRC users that were active only on this IRC channel. The ratio is calculated from IRC users who have joined the channel.

The next subsection is focused on describing the mean and STD of Levenshtein distance of IRC messages.

6.3.1.1 Pairwise Levenshtein Distance between IRC Messages

To reflect how much the messages diversify in the IRC channels, we compare Levenshtein Distance (LD) of all messages pair-wisely. Levenhstein Distance is a metric for measuring the number of differences between two strings. The LD is defined as follows:

$$LD(s,t) \tag{6.1}$$

where s is a source string and t is a target string. The distance between strings is measured by the number of insertion, deletion or substitution required to transform string s to string t. Therefore, the higher the LD between all messages, the more different the messages are, and the less likely that the IRC channel contains messages that are repetitively sent as command-and-control commands. We calculated the LD for each possible pair of messages and calculated mean and standard deviation from the resulting values.

6.3.2 Features Visualization

We projected the feature vectors into lower dimensions. To project the feature vectors, the same methods were used as in Chapter 5. Firstly, we applied the Principal Component Analysis (PCA) 6.1. Then we projected the data using the t-SNE 6.2 and the Uniform Manifold Approximation (UMAP) 6.3 algorithms. Visualization showed that malicious IRC channels are projected close to each other, but no clear separation of benign IRC channels and malicious IRC channels are visible from the visualizations.

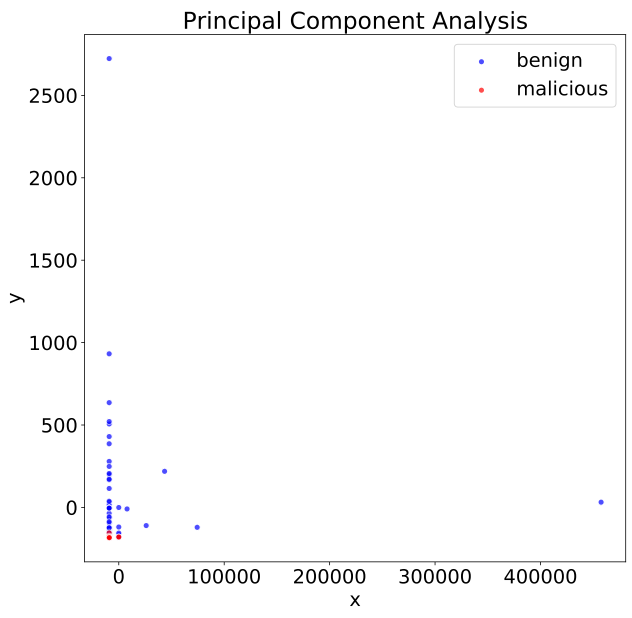

Figure 6.1: PCA visualization of IRC channels in the IRC Channel & User dataset. Red dots represent malicious IRC channels, and blue dots represent benign IRC channels.

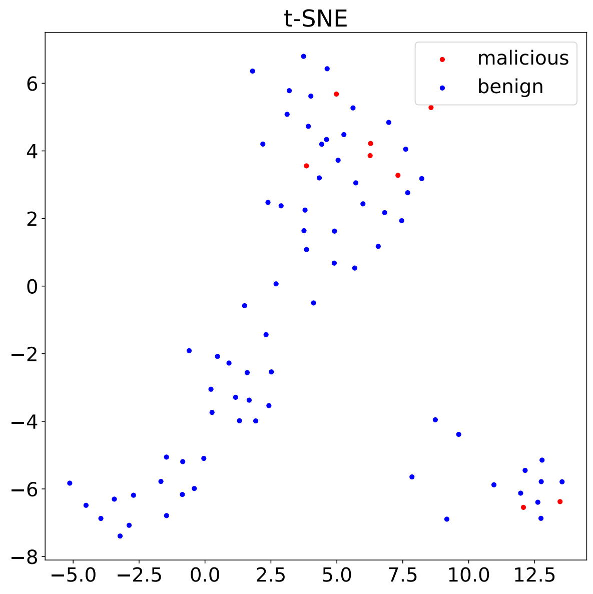

Figure 6.2: t-SNE visualization of IRC channels in the IRC Channel & User dataset. Red dots represent malicious IRC channels, and blue dots represent benign IRC channels.

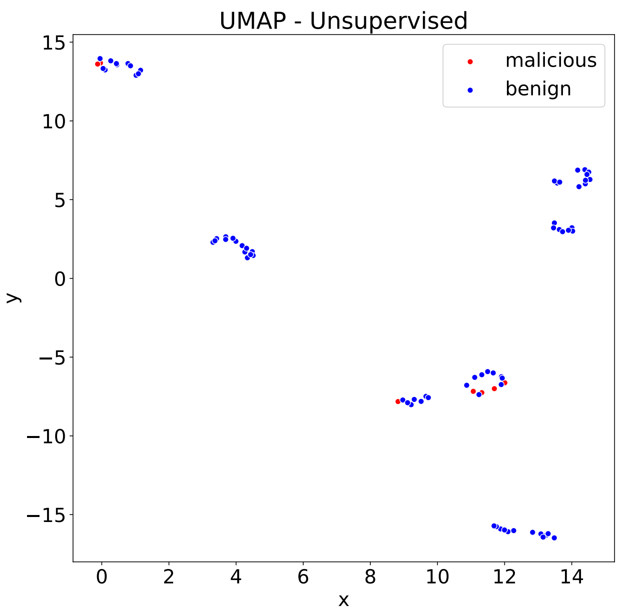

Figure 6.3: UMAP visualization of IRC channels in the IRC Channel & User dataset. Red dots represent malicious IRC channels, and blue dots represent benign IRC channels.

6.3.3 Results

Results in Table 6.3 shows that unsupervised methods perform poorly on the problem of detecting malicious IRC channels in network traffic, as both unsupervised methods achieved an F1-score of 0%. Among supervised methods, Linear Classifier had the worst performance and achieved an F1-score of 33.33%. Logistic Regression achieved an F1-score of 33.33%. SVM achieved an F1-score of 57.14%. All methods were outperformed by Random Forest that achieved a F1-score of 80.00%. The last evaluated method XGboost achieved the F1-score of 40.00%.

In Figure 6.4, the learning curve of Random Forest is visualized. The figure shows that an increasing number of samples, both train F1-score and test F1-score, decrease. However, when the number of samples gets higher, the model starts performing well on both sets of samples, and the training error and the test error become smaller. Learning curves for the rest of the evaluated models are attached in Appendix A.

6.3.3.1 Feature Importance

We interpreted the evaluated models to see what are the most important features. In Figure 6.5, the most important features of the Random Forest Classifier are visualized. The most important feature of the final model is the mean of message special characters. The final decision is made together with another five features: maximal number of joined users, message periodicity, standard deviation of pairwise Levenshtein distance between messages,

Table 6.3: IRC Channel Detection: Score of machine learning methods trained on the imbalanced IRC Channel&User dataset.

Method	Hyper-parameters	TN	FN	TP	FP	Accuracy	F1-score	FPR
K-Means	k: 3	72	1	0	8	88.89%	0%	10.00%
Hierarchical Clustering	-	72	1	0	8	88.89%	0%	10%
Linear Classifier	-	28	2	1	2	87.88%	33.33%	6.67%
Logistic Regression	-	30	0	1	2	93.94%	50.00%	6.25%
SVM	kernel: RBF $C : 0.1$ $\gamma : 0.1$	28	2	2	1	90.91%	57.14%	3.45%
Random Forest	estimators: 10 tree max depth: 4	30	0	2	1	96.97%	80.00%	3.23%
XGBoost	estimators: 5 tree max depth: 2 γ: 0 α: 0.01 λ: 0.01	29	1	1	2	90.91%	40.00%	6.45%

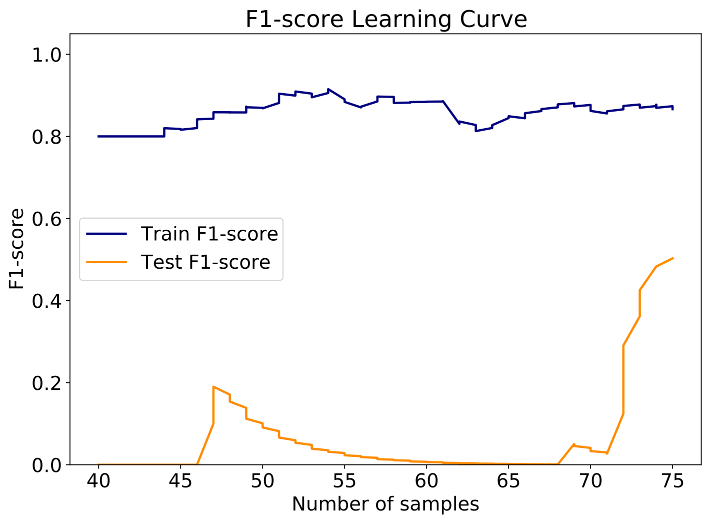

Figure 6.4: IRC Channel Detection: Learning Curve of Random Forest Classifier

message word entropy, and number of active users. On the contrary, one-channel user ratio, message frequency, mean of pairwise Levenshtein distance, and language did not have a big importance for the final decision.

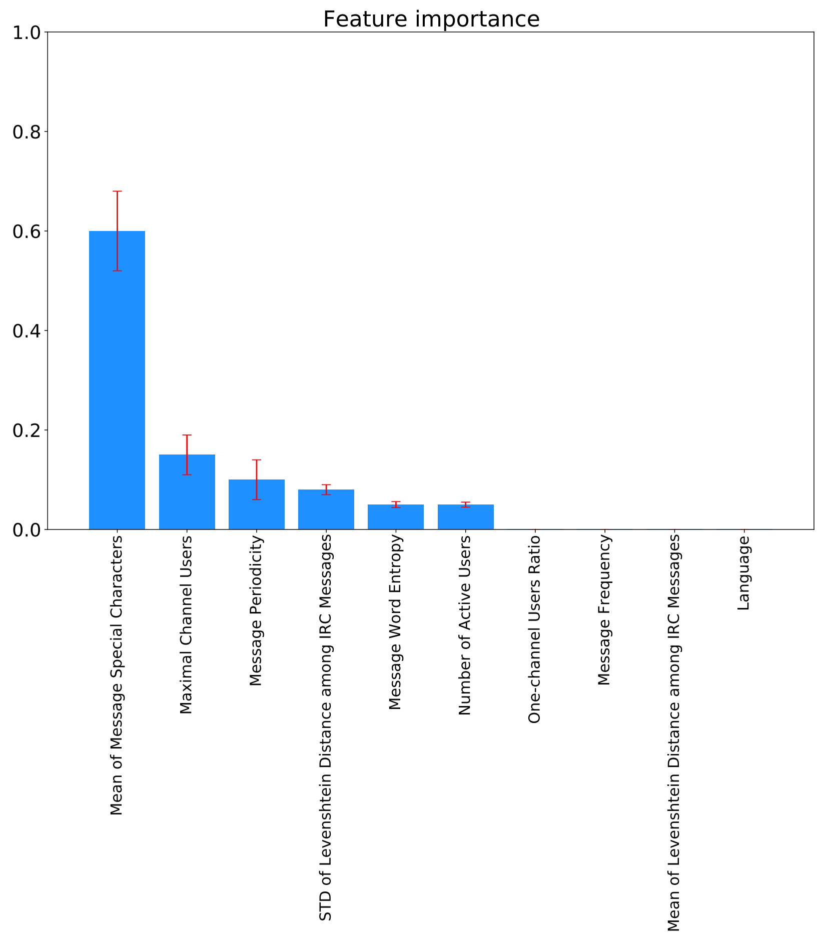

Figure 6.5: IRC Channel Detection: Feature Importance of Random Forest Classifier

6.3.4 Class Balancing

To balance classes in the IRC Channel & User dataset, we performed undersampling and oversampling. By undersampling the IRC Channel & User dataset, a random subset of malicious samples was selected to match the number of benign samples. On the contrary, by oversampling the IRC Channel & User dataset, random subset of malicious samples is duplicated to match the number of samples in the benign class. Description of the undersampled IRC Channel & User dataset and oversampled IRC Channel & User dataset is shown in Table 6.4 and Table 6.5, respectively.

Table 6.4: IRC Channel Detection: Description of the undersampled IRC Channel & User dataset

Type of IRC	IRC channels	IRC users	IRC messages
Malicious	8	19	7,643
Benign	8	60	1,608

Table 6.5: IRC Channel Detection: Description of the oversampled IRC Channel & User dataset

Type of IRC	IRC channels	IRC users	IRC messages
Malicious	73	173	69,742
Benign	73	563	99,250

6.3.4.1 Evaluation

We trained detection models on the undersampled dataset and oversampled dataset to verify the correctness of our results achieved on the original IRC Channel & User dataset. The evaluation process remains the same as described in Section 5.3.1.

The evaluation results of detection models trained on the undersampled IRC Channel & User dataset are shown in Table 6.6. We can notice that the F1-score of all detection models improved except the Random Forest Classifier that achieved the F1-score of 40.00%. XGBoost performed the best among the other methods and achieved the F1-score of 85.71%. Among the unsupervised methods, both methods, K-Means and Hierarchical Clustering, improved and achieved the F1-score of 57.14% and 36.36%, respectively. For the rest of the supervised methods, Linear Classifier together with SVM, improved by achieving the F1-score of 33.33% both. Logistic Regression did not improved and achieved the F1-score of 57.14.

The evaluation results of detection models trained on the oversampled dataset are shown in Table 6.7. Oversampling did not improve an F1-score of the unsupervised methods. Both methods, K-Means and Hierarchical Clustering, achieved a zero F1 score. For the supervised methods, the perfect F1-score of 100.00 was achieved by SVM and XGBoost. The worst F1-score among the supervised methods was achieved by Linear Classifier that achieved the F1-score of 33.33%. Other supervised methods, Logistic Regression and Random Forest, achieved the F1-score of 78.12% and 96.55%, respectively.

Undersampling the original dataset worsened the results for the majority of the methods. That is caused by having a lack of data samples in the minority class of the original dataset.

Table 6.6: IRC Channel Detection: Score of machine learning methods trained on undersampled IRC Channel & User dataset

Method	Hyper-parameters	TN	FN	TP	FP	Accuracy	F1-score	FPR
K-Means	k: 3	1	7	6	2	43.75%	57.14%	66.67%
Hierarchical Clustering	-	7	1	2	6	56.25%	36.36%	46.15%
Linear Classifier	-	2	1	1	3	42.86%	33.33%	60.00%
Logistic Regression	-	2	1	2	2	57.14%	57.14%	50.00%
SVM	kernel: RBF $C : 0.1$ $\gamma : 0.1$	2	1	1	3	42.86%	33.33%	60.00%
Random Forest	estimators: 10 tree max depth: 4	3	0	1	3	57.14%	40.00%	50.00%
XGBoost	estimators: 5 tree max depth: 2 γ: 0 α: 0.01 λ: 0.01	3	0	3	1	85.71%	85.71%	25.00%

Table 6.7: IRC Channel Detection: Score of machine learning methods trained on oversampled IRC Channel & User dataset

Method	Hyper-parameters	TN	FN	TP	FP	Accuracy	F1-score	FPR
K-Means	k: 4	72	1	0	73	49.32%	0%	50.34%
Hierarchical Clustering	-	72	1	0	73	49.32%	0%	50.34%
Linear Classifier	-	22	7	21	9	72.88%	72.41%	29.03%
Logistic Regression	-	20	9	25	5	76.27%	78.12%	20.00%
SVM	kernel: RBF $C : 0.1$ $\gamma : 0.1$	29	0	30	0	100.00%	100.00%	0%
Random Forest	estimators: 10 tree max depth: 4	29	0	28	2	96.61%	96.55%	6.45%
XGBoost	estimators: 5 tree max depth: 2 γ: 0 α: 0.01 λ: 0.01	29	0	30	0	100.00%	100.00%	0%

6.4 IRC User Detection

IRC users meet and chat between themselves privately via IRC channels. In IRC-based botnets, botmasters and bots are represented as IRC users to be able to communicate between themselves. This section describes a method used for detection of IRC users that are part of botnets. We separated the whole network traffic per IRC users. For every IRC user, features were extracted from the network traffic and used as a data input into detection models.

6.4.1 Feature Selection

In this section, we describe the complete list of features that were extracted for each IRC user. The features were chosen to provide us a meaningful representation of the IRC User biased towards separating malicious IRC users from benign IRC users. The features we extract for each IRC user are:

- **Message Periodicity** - We suppose that bots that are controlled by a botmaster use IRC for sending commands periodically, so we wanted to obtain the value of periodicity. To do that, we created a method that would return a number between 0 and 1 - i.e. one if the message sequence is perfectly periodical, zero if the message sequence is not periodical at all. The computation process of the feature was described in Section 5.2.1.
- **Mean of Username Special Characters** - To identify whether the username of the IRC user is randomly generated or not. Therefore, in this feature, we compute the average usage of non-alphabetic characters in the username. The computation process of the feature was described in Section 5.2.3.
- **Mean of Message Special Characters** - If the message contains many commands, it will most likely contain a lot of special characters than the ordinary message. With this feature, we obtain the mean usage of non-alphabetic characters across all sent messages by the IRC user. The computation process of the feature was described in Section 5.2.4.
- **Message Word Entropy** - To consider whether the IRC user sends the same message multiple times in a row, or whether the message contains a limited number of words, we compute a word entropy across all of the IRC messages sent by the IRC user. The computation process of the feature was described in Section 5.2.2.
- **Message Frequency** - Frequency of IRC messages sent by the IRC user. Computed as a time of a network capture divided by a number of IRC messages sent by the IRC user.
- **Mean of Levenshtein Distance of IRC Messages** - Mean of pairwise Levenshtein distance of IRC messages sent by the IRC user. The computation process of the feature was described in Section 6.3.1.1.
- **STD of Levenshtein Distance of IRC Messages** - Standard deviation (STD) of pairwise Levenshtein distance of IRC messages sent by the IRC user. The computation process of the feature was described in Section 6.3.1.1.
- **User's Language** - Recognized language from IRC messages sent by the IRC user.
- **Number of Joined Channels** - Number of channels that the IRC user joined. IRC Users that joined a channel see the content of the channel.
- **Number of Active Channels** - Number of channels where the IRC user sent at least one message during the traffic capture.
- **Number of Nickname Changes** - Number of nickname changes made by the IRC user during the traffic capture.

6.4.1.1 Data Visualization

We projected the feature vectors into lower dimensions. To project the feature vectors, we applied Principal Component Analysis (PCA) 6.6. Then we projected the data using t-SNE 6.7 and Uniform Manifold Approximation (UMAP) 6.8 algorithms. Visualization showed that the majority of malicious IRC users are projected close to each other, but no clear separation of benign IRC users and malicious IRC users are visible from the visualization.

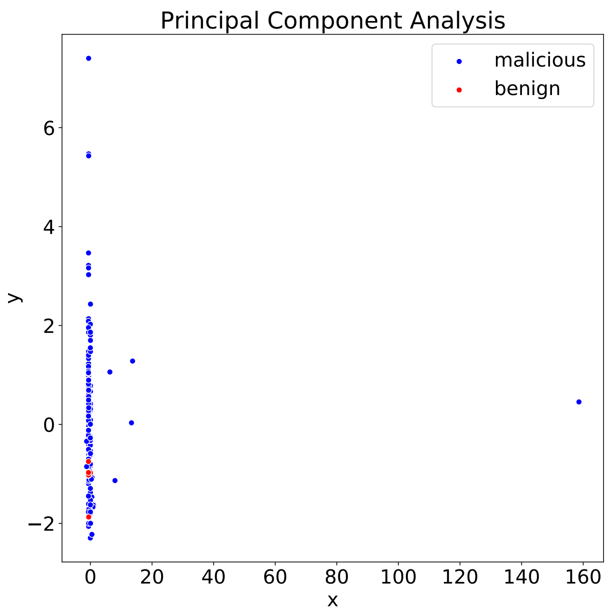

Figure 6.6: PCA visualization of IRC users in the IRC Channel & User dataset. Red dots represent malicious IRC users, and blue dots represent benign IRC users.

6.4.2 Results

Results in Table 6.8 showed us that unsupervised methods do not show a good performance, and both unsupervised methods, K-Means and Hierarchical Clustering, achieved the F1-score of 0%.

Among supervised methods, The worst performance had Logistic Regression and SVM that both achieved the F1-score of 0%. Linear Classifier did not perform well and achieved the F1-score of 2.90%. SVM achieved the F1-score of 93.33%. The best-performed model is the Random Forest Classifier that achieved the F1-score of 90.91%. XGBoost achieved the F1-score of 28.57%.

In Figure 6.9, the learning curve of Random Forest Classifier is visualized. The figure shows that with increasing the number of samples, the training error becomes larger. However, when the model is trained on more data samples, it manages to fit the test set better. While the test F1-score still increases with a higher number of samples, a larger data set

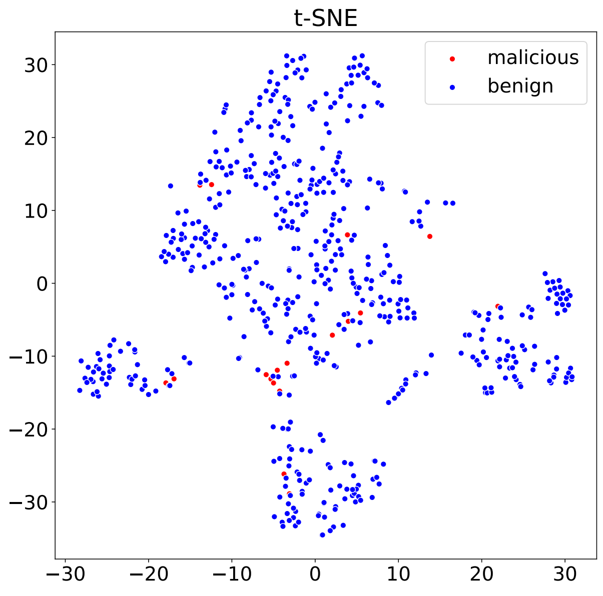

Figure 6.7: t-SNE visualization of IRC users in the IRC Channel & User dataset. Red dots represent malicious IRC users, and blue dots represent benign IRC users.

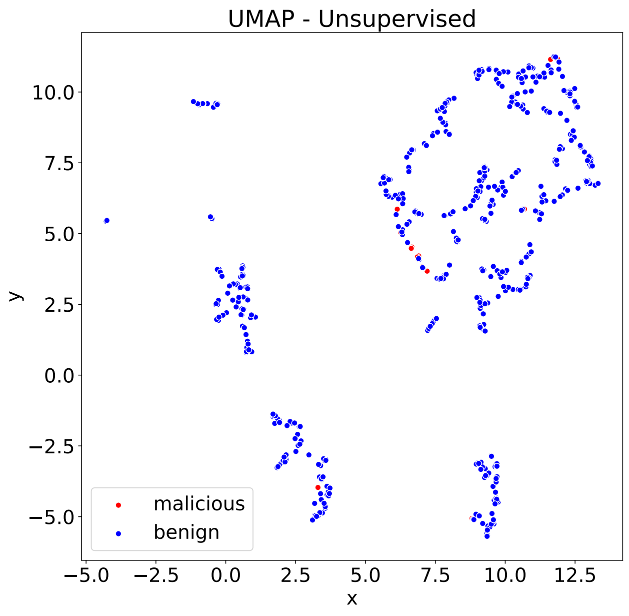

Figure 6.8: UMAP visualization of IRC users in the IRC Channel & User dataset. Red dots represent malicious IRC users, and blue dots represent benign IRC users.

would improve the resulting model quality. Learning curves for the rest of the evaluated models are attached in Appendix A.

Table 6.8: IRC User Detection: Score of machine learning methods trained on the imbalanced IRC Channel & User dataset

Method	Hyper-parameters	TN	FN	TP	FP	Accuracy	F1-score	FPR
K-Means	k: 3	562	1	0	19	96.56%	0%	3.27%
Hierarchical Clustering	-	562	1	0	19	96.56%	0%	3.27%
Linear Classifier	-	1	5	107	62	61.71%	76.16%	98.41%
Logistic Regression	-	0	6	169	0	96.57%	98.26%	-
SVM	kernel: RBF C : 0.01 γ : 0.1	0	6	169	0	96.57%	98.26%	-
Random Forest	estimators: 10 tree max depth: 4	5	1	169	0	99.43%	90.91%	0%
XGBoost	estimators: 5, tree max depth: 2 γ: 0, α: 0.01 λ: 0.01	1	5	169	0	97.14%	28.57%	0%

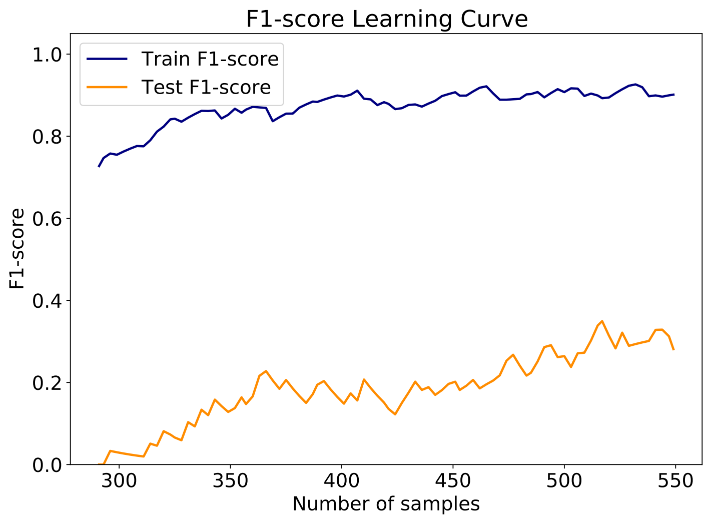

Figure 6.9: IRC User Detection: Learning Curve of Random Forest Classifier

6.4.2.1 Feature Importance

We interpreted evaluated models to see what are the most important features. In Figure 6.5, the most important features of the Random Forest Classifier are visualized. The most important feature of the final model is the Mean of Message Special Characters. The final decision is made together with the rest of the features except one feature: Number of Nickname Changes.

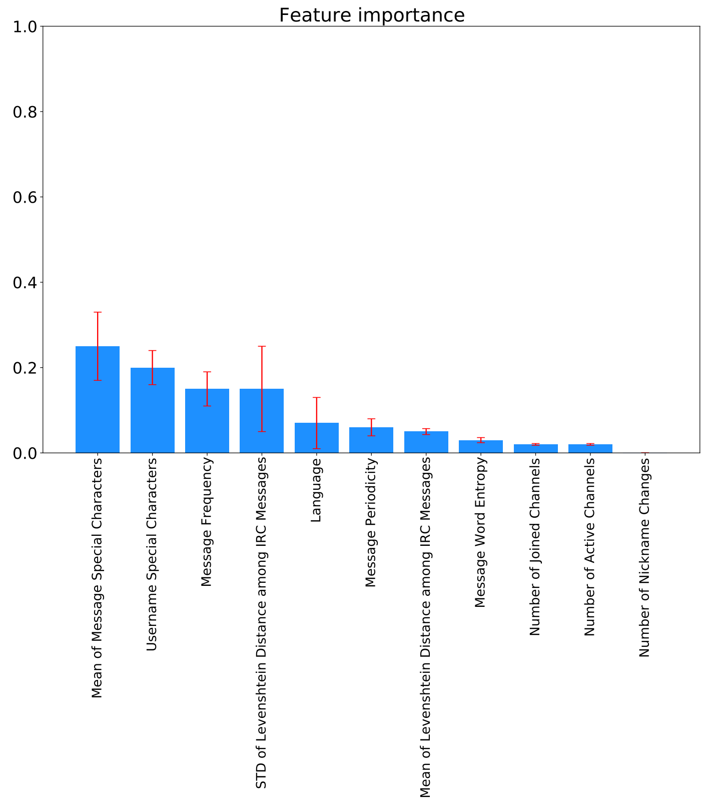

Figure 6.10: IRC User Detection: Feature Importance of Random Forest Classifier

6.4.3 Class Balancing

To balance classes in the IRC Channel & User dataset, we performed undersampling and oversampling. By undersampling the IRC Channel & User dataset, a random subset of malicious samples was selected to match the number of benign samples. On the contrary, by oversampling the IRC Channel & User dataset, a random subset of malicious samples is duplicated to match the number of samples in the benign class. Description of the undersampled IRC Channel & User dataset and oversampled IRC Channel & User dataset is shown in Table 6.9 and Table 6.10, respectively.

Table 6.9: IRC User Detection: Description of the undersampled IRC Channel & User dataset

Type of IRC	IRC channels	IRC users	IRC messages
Malicious	8	19	7,643
Benign	2	19	3,310

Table 6.10: IRC User Detection: Description of the oversampled IRC Channel & User dataset

Type of IRC	IRC channels	IRC users	IRC messages
Malicious	210	563	16,327
Benign	73	563	99,250

6.4.3.1 Evaluation

We trained detection models on balanced datasets to verify the correctness of our results achieved on the original IRC Channel & User dataset. The evaluation process remains the same as described in Section 5.3.1.

The evaluation results of detection models trained on the undersampled dataset are shown in Table 6.11. We can notice that an F1-score of both unsupervised methods, K-Means and Hierarchical Clustering, improved by achieving the F1-score of 17.14% and 20.00%, respectively. Among the supervised methods, Random Forest and XGBoost performed the best and achieved the F1-score of 66.67%. On the contrary, the worst-performing method was Linear Classifier, where the F1-score of 46.15% has been achieved. The rest of the supervised methods, Logistic Regression and SVM, achieved the F1-score of 61.54% and 57.14%, respectively.

The evaluation results of detection models trained on the oversampled dataset are shown in Table 6.12. It is apparent that oversampling did not improve an F1-score of the unsupervised methods, and both methods, K-Means and Hierarchical Clustering achieved the F1-score of 0%. Among supervised methods, the perfect F1-score of 100.00% was achieved by Random Forest Classifier. The worst performing supervised method was Linear Classifier that achieved the F1-score of 75.44%. Compared to the evaluation methods on the imbalanced dataset, the rest of the supervised methods performed better. Logistic Regression achieved the F1-score of 82.39%. SVM and XGBoost achieved the F1-score of 95.48% and 99.38%, respectively.

By applying the class balancing methods, the results have improved for undersampling the dataset. On the contrary, oversampling the dataset did not perform well. That is caused by a low amount of data samples in the minority class of the imbalanced IRC Channel & User dataset.

Table 6.11: IRC User Detection: Score of machine learning methods trained on undersampled IRC Channel & User dataset

Method	Hyper-parameters	TN	FN	TP	FP	Accuracy	F1-score	FPR
K-Means	k: 3	6	13	3	16	23.68%	17.14%	72.73%
Hierarchical Clustering	-	11	8	3	16	36.84%	20.00%	59.26%
Linear Classifier	-	2	4	3	3	41.67%	46.15%	60.00%
Logistic Regression	-	3	3	4	2	58.33%	61.54%	40.0%
SVM	kernel: RBF C : 0.01 γ : 0.1	2	4	4	2	50.00%	57.14%	50.0%
Random Forest	estimators: 10 tree max depth: 4	5	1	2	3	72.73%	50.00%	37.5%
XGBoost	estimators: 5 tree max depth: 2 γ: 0 α: 0.01 λ: 0.01	3	2	4	2	63.64%	66.67%	40.0%

Table 6.12: IRC User Detection: Score of machine learning methods trained on oversampled IRC Channel & User dataset

Method	Hyper-parameters	TN	FN	TP	FP	Accuracy	F1-score	FPR
K-Means	k: 4	562	1	0	563	49.91%	0%	50.04%
Hierarchical Clustering	-	562	1	0	563	49.91%	0%	50.04%
Linear Classifier	-	149	20	106	63	75.44%	71.86%	29.72%
Logistic Regression	-	141	28	138	31	82.54%	82.39%	18.02%
SVM	kernel: RBF C : 0.01 γ : 0.1	153	16	169	0	95.27%	95.48%	0%
Random Forest	estimators: 10 tree max depth: 4	169	0	169	0	100.00%	100.00%	0%
XGBoost	estimators: 5 tree max depth: 2 γ: 0 α: 0.01 λ: 0.01	158	2	160	0	99.38%	99.38%	0%

Chapter 7

Comparison with State-of-the-art IDSs

This chapter compares the efficiency of proposed methods with the state-of-the-art Intrusion Detection Systems (IDSs), Snort and Suricata [5, 6]. The comparison is performed by partitioning the collected network traffic per IRC 3-tuple sessions, IRC channels, and IRC users. For each divided network traffic, the detection is performed for all compared detectors, i.e., our proposed detector, Snort, and Suricata. Finally, the detectors' efficiency is compared together, and the resulting score is computed and discussed.

7.1 Comparison Methodology

Unlike our created methods, Snort and Suricata monitor the whole network traffic at once. Therefore, to verify the soundness of the proposed methods against IDSs, it is necessary to split the network traffic. We split the collected network traffic per IRC 3-tuple sessions, IRC channels, and IRC users. The description of the collected network traffic used for the comparison is shown in Table 7.1. The collected network traffic is the same as in Chapter 6. Table 7.2 lists the dataset description used in our experiments.

Table 7.1: Description of the network traffic used to compare created detection methods against the state-of-the-art IDS Snort and Suricata.

Scenario	IRC 3-Tuple Sessions	IRC Channels	IRC Users
CTU-IoT-Malware-Capture-34	53	1	6
CTU-IoT-Malware-Capture-42	1	1	1
CTU-IoT-Malware-Capture-51	2	3	3
CTU-IoT-Malware-Capture-56	1	2	6
CTU-IoT-Malware-Capture-62	1	1	3
University Benign IRC Traffic	1,470	73	563

Table 7.2: Description of the dataset used to compare created detection methods against the state-of-the-art IDS Snort and Suricata.

Type of IRC	IRC 3-tuple sessions	IRC Channels	IRC Users
Malicious	56	8	19
Benign	1,470	73	563

7.1.1 IDS Evaluation

Snort and Suricata are rule-based IDS. It means that both systems, Snort and Suricata, require a set of rules for their evaluation. The rule action tells Snort and Suricata what to do when it finds a packet that matches a pattern described in the rule criteria. An alert is usually the output of a rule action. For the comparison purposes, we used the default rule sets in both IDSs Snort, and Suricata. The evaluated versions of Snort and Suricata were `5.0.3` with the rule set version 2983 and `2.9.16` with the the rule set version `9543`, respectively.

Snort analysis of network traffic is executed by using the command below:

```
#!/bin/bash
snort -c <snort_config>.conf -r <file.pcap> -A console
```

where `<snort_config>` refers to a config file with a rule set, `<file.pcap>` refers to a file of a traffic capture that is being analyzed, and the switch `-A console` ensures that all alarm logs are shown in the console. An example of the console output is shown in Appendix B.

Suricata analysis of network traffic is executed by using the command below:

```
#!/bin/bash
suricata -c suricata_config.yaml -r <file>.pcap -l <out_logs>
```

where `suricata_config.conf` refers to a config file with a rule set, `<file>` refers to a traffic capture file that is being analyzed, and `<out_logs>` refers to an output log directory. The example output of Suricata is shown in Appendix B.

Alert logs were used to identify that Snort and Suricata recognized malware in network traffic. There are several alert types and alert priorities. To identify malware, we used a malware classification log with priority 1. If this alert occurs in IDS output alerts, then we determine that IDS identified malware in the network traffic. An example of the alert that is used to determine malware detection is shown below:

```
05/21/2019-03:29:03.583121  [**] [1:2000347:16] ET MALWARE IRC Private
↪  message on non-standard port [**] [Classification: A Network Trojan was
↪  detected] [Priority: 1] {TCP} 192.168.1.196:43736 -> 149.56.41.138:80
```

7.2 IRC 3-Tuple Session Detection

Detection of malicious IRC 3-tuple sessions was performed by splitting the collected network traffic into IRC 3-Tuple sessions which are defined by source IP address, source port, and destination IP address. The network traffic partition into IRC 3-tuple sessions was done by using the command bellow:

```
#!/bin/bash
tcpdump -tttt -n --number -s0 -r <file_in>.pcap -A host <src_ip>
and <dst_ip> and port <dst_port> -w <file_out>.pcap
```

where `<src_ip>` represents the source IP, `<dst_ip>` represents the destination IP, and `<dst_port>` refers to the destination port of the particular IRC 3-tuple session, `<file_in>` and `<file_out>` refers to an input network capture file, and an output network capture file, respectively.

The results of all detectors executed on a partitioned network traffic per 3-tuple sessions are shown in Table 7.3. The results show that our proposed detector overperformed both IDSs, achieving the F1-score of 98.18%. Snort and Suricata achieved the F1-score of 90.20% and 92.31%, respectively.

Detector	TN	FN	TP	FP	Accuracy	F1-Score	FPR
IRC 3-Tuple Session Detector	1470	2	54	0	99.87%	98.18%	0%
Snort	1470	10	46	0	99.34%	90.20%	0%
Suricata	1470	8	48	0	99.48%	92.31%	0%

Table 7.3: Comparison of IRC 3-Tuple Session Detection against the state-of-the-art IDS Snort and Suricata.

7.3 IRC Channel Detection

Detection of malicious IRC channels was performed by splitting the collected network traffic into IRC channels. The network traffic partition into IRC 3-tuple channels was done by using the command below:

```
#!/bin/bash
tshark -nr  <file_in.pcap> -Y 'frame contains "<#channel_name>"'
-T fields -e ip.src -e ip.dst -e tcp.srcport -e tcp.dstport |
uniq | awk '{if ( $1 == "<client_ip>") print "tcpdump -n -s0 -r
<file_in>.pcap> host "$1" and port "$2" and host "$3" and port
"$4" -v -w <file_out>.pcap"}'
```

where `<#channel_name>` refers to the filtered channel name, `<client_ip>` refers to the client IP address, `<file_in>` and `<file_out>` represent the input network capture file, and output network capture file, respectively. This command is composed of multiple subcommands. `tshark` command returns a list of tuples. The tuple consists of a source IP address, destination IP address, and destination port. The list of tuples identifies all users that communicate in the filtered channel. Finally, the duplicates in the tuple list were removed and the resulting list was used as an input to `tcpdump` command. This command filters network traffic by particular users who communicate within the channel.

The results of all detectors executed on a partitioned network traffic per channels are shown in Table 7.4. Our proposed detector significantly overperformed both IDSs, achieving the F1-score of 98.18%. Snort did not detect any of the malicious IRC channels and achieved zero F1-score. Suricata performed better and achieved the F1-score of 40.00%.

Detector	TN	FN	TP	FP	Accuracy	F1-Score	FPR
IRC Channel Detector	72	1	7	1	97.53%	87.5%	1.37%
Snort	73	8	0	0	90.12%	0%	0%
Suricata	73	6	2	0	92.59%	40.00%	0%

Table 7.4: Comparison of IRC Channel Detection against the state-of-the-art IDS Snort and Suricata

7.4 IRC User Detection

Detection of malicious IRC users was performed by splitting the collected network traffic into IRC users. The network traffic partition into IRC users was done by using the command below:

```
#!/bin/bash
tshark -nr  <file_in.pcap> -Y 'frame contains "<user@domain>"'
-T fields -e ip.src -e ip.dst -e tcp.srcport -e tcp.dstport |
uniq | awk '{if ( $1 == "<client_ip>") print "tcpdump -n -s0 -r
<file_in>.pcap> host "$1" and port "$2" and host "$3" and port
"$4" -v -w <file_out>.pcap"}'
```

This command filters network traffic for the particular user that is represented by the `<user@domain>`. This command consists of multiple subcommands, and it is described more in detail in Section 7.3.

The results of all detectors executed on a partitioned network traffic per user are shown in Table 7.5. The results show that only our proposed detector of IRC users successfully detects any of the malicious IRC users. Both IDSs, Snort and Suricata, did not detect any of the malicious IRC users, and both IDSs achieved the zero F1-score. The filtered traffic of the particular user may not contain enough traffic to be detected by IDS systems.

Detector	TN	FN	TP	FP	Accuracy	F1-Score	FPR
IRC User Detector	563	5	14	0	99.14%	84.85%	0%
Snort	563	19	0	0	96.74%	0%	0%
Suricata	563	19	0	0	96.74%	0%	0%

Table 7.5: Comparison of IRC User Detection against the state-of-the-art IDS Snort and Suricata

7.5 Comparison Summary

The comparison shows that all proposed detection methods are sound and outperformed state-of-the-art IDS Snort and Suricata. The comparison results show that IDS systems require more complex network traffic. The network traffic of the particular user is not enough to detect user's malicious attempts. On the contrary, our proposed detector of malicious IRC channels classified false-positively one benign channel, and both IDS systems achieved zero False Positive Rate (FPR) in all experiments. That shows that IDS systems prefer not to generate false-positive alerts in exchange for a higher false-negative rate.

Chapter 8

Conclusion

We have proposed a new approach to split the IRC communication into IRC 3-tuple sessions. The IRC 3-tuple session is defined as a group of flows between the source IP, destination IP, and destination port. Data visualization was used to explore selected features as a suitable representation of the model since malicious IRC 3-tuple sessions were projected close to each other. The results showed that botnets could be accurately detected by our approach of extracting IRC 3-tuple sessions from network traffic. Not all of the unsupervised methods were suitable for the detection of malicious IRC communication. K-Means did not perform well. However, Hierarchical Clustering achieved an F1-score of 92.68%. For supervised methods, the results showed that the linear models could not capture the complexity of the data. Random Forest and XGBoost models outperformed the rest of the methods and achieved an F1-score of 95.24%. Linear Classifier and Logistic Regression were not accurate.

The other proposed approach was to profile IRC communication per IRC channels and per IRC users. Results of detecting malicious IRC channels showed that unsupervised methods do not show a good performance on our problem, and both unsupervised methods achieved the F1-score of 0%. Among supervised methods, all methods were outperformed by the Random Forest Classifier that achieved the F1-score of 80.00%. Results of detecting malicious IRC users in network traffic showed us that unsupervised methods do not show a good performance, and both unsupervised methods achieved the F1-score of 0%. Among supervised methods, the best-performing model was the Random Forest Classifier that achieved an F1-score of 90.91%. The learning curve of the Random Forest Classifier showed us that the test F1-score still increases with a higher number of samples. Therefore a larger data set would improve the resulting model quality.

The evaluation of all machine learning methods showed us that Random Forrest Classifier is the most stable model in terms of performance across all the experiments. Random Forrest Classifier outperformed the rest of the models in the majority of experiments, especially for the models trained on imbalanced datasets and the models trained on oversampled datasets. In terms of class balancing methods, undersampling did not work well in general on our imbalanced datasets. That is caused by having a lack of data samples in the minority class of the datasets. Finally, a comparison against the state-of-the-art IDS Snort and Suricata was done to verify that our technique was sound. The comparison results showed that all proposed detection methods are sound and outperformed state-of-the-art IDS, Snort, and

Suricata. The comparison results showed that IDS systems prefer not to generate false-positive alerts in exchange for a higher false-negative rate.

In future work, we are planning to integrate the detection of malicious IRC channels and IRC users into SLIPS detection system. By having all three proposed detection methods integrated, we will be able to combine the outputs from detection methods. The combination of outputs from all detection methods will help us to detect and profile the botnets even more precisely. The next topic to cover is the analysis of the Peer to Peer (P2P) connection. P2P is used inside the IRC protocol for direct communication between users. Since the proposed methods for detection of malicious IRC 3-tuple sessions, IRC Channels, and IRC users are now focusing only on content in channels, the impact can be increased by extending work with a model that would detect and analyze malicious P2P communication.

Bibliography

[1] About QuakeNet, . Retrieved April 4, 2020, from <http://www.quakenet.org/about>.

[2] IRC Networks 2003 - Top 10 in the annual comparison, . Retrieved April 5, 2020, from <www.netsplit.de/networks/top10.php?year=2003>.

[3] IRC Networks 2019 - Top 10 in the annual comparison, . Retrieved April 5, 2020, from <www.netsplit.de/networks/top10.php?year=2019>.

[4] Stratosphere Linux IPS. Retrieved June 5, 2020, from <https://www.stratosphereips.org/stratosphere-ips-for-linux>.

[5] SNORT. Retrieved June 5, 2020, from <https://www.snort.org>.

[6] Surricata. Retrieved July 10, 2020, from <https://suricata-ids.org>.

[7] VirusTotal. Retrieved June 5, 2020, from <https://www.virustotal.org>.

[8] Wireshark. Retrieved July 10, 2020, from <https://www.wireshark.org>.

[9] Zeek. Retrieved June 5, 2020, from <https://www.zeek.org>.

[10] AGUSTIN PARMISANO, M. J. E. S. G. Aposemat IoT-23. Stratosphere Laboratory. A labeled dataset with malicious and benign IoT network traffic. Retrieved January 22, 2020, from <https://www.stratosphereips.org/datasets-iot23>.

[11] PRAGUE, S. L. Aposemat IoT Project. Retrieved May 30, 2020, from <https://www.stratosphereips.org/aposemat>.

[12] BARSAMIAN, A. V. *Network characterization for botnet detection using statistical-behavioral methods*. PhD thesis, Citeseer, 2009.

[13] BINKLEY, J. R. – SINGH, S. An algorithm for anomaly-based botnet detection. *SRUTI*. 2006, 6, s. 7–7.

[14] CHOI, H. et al. Botnet detection by monitoring group activities in DNS traffic. In *7th IEEE International Conference on Computer and Information Technology (CIT 2007)*, s. 715–720. IEEE, 2007.

[15] CHOI, H. – LEE, H. – KIM, H. BotGAD: detecting botnets by capturing group activities in network traffic. In *Proceedings of the Fourth International ICST Conference on COMmunication System softWAre and middlewaRE*, s. 1–8, 2009.

[16] COOKE, E. – JAHANIAN, F. – MCPHERSON, D. The Zombie Roundup: Understanding, Detecting, and Disrupting Botnets. *SRUTI*. 2005, 5, s. 6–6.

[17] DE DONNO, M. et al. DDoS-capable IoT malwares: Comparative analysis and Mirai investigation. *Security and Communication Networks*. 2018, 2018.

[18] GOEBEL, J. – HOLZ, T. Rishi: Identify Bot Contaminated Hosts by IRC Nickname Evaluation. *HotBots*. 2007, 7, 8-8, s. 192.

[19] GU, G. et al. Bothunter: Detecting malware infection through ids-driven dialog correlation. In *USENIX Security Symposium*, 7, s. 1–16, 2007.

[20] GU, G. et al. Botminer: Clustering analysis of network traffic for protocol-and structure-independent botnet detection. 2008.

[21] GU, G. – ZHANG, J. – LEE, W. BotSniffer: Detecting botnet command and control channels in network traffic. 2008.

[22] GU, G. et al. Active botnet probing to identify obscure command and control channels. In *2009 Annual Computer Security Applications Conference*, s. 241–253. IEEE, 2009.

[23] KOLIAS, C. et al. DDoS in the IoT: Mirai and other botnets. *Computer*. 2017, 50, 7, s. 80–84.

[24] LIVADAS, C. et al. Using machine learning techniques to identify botnet traffic. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, s. 967–974. IEEE, 2006.

[25] LIVINGOOD J., M. M. Recommendations for the Remediation of Bots in ISP Networks. RFC 6561, 2012. Retrieved April 5, 2020, from <https://tools.ietf.org/html/rfc6561>.

[26] MARZANO, A. et al. The evolution of bashlite and mirai iot botnets. In *2018 IEEE Symposium on Computers and Communications (ISCC)*, s. 00813–00818. IEEE, 2018.

[27] MASUD, M. M. et al. Flow-based identification of botnet traffic by mining multiple log files. In *2008 First International Conference on Distributed Framework and Applications*, s. 200–206. IEEE, 2008.

[28] MAZZARIELLO, C. IRC traffic analysis for botnet detection. In *2008 The Fourth International Conference on Information Assurance and Security*, s. 318–323. Ieee, 2008.

[29] MICHAEL, M. Attack Landscape H1 2019: IoT, SMB traffic abound, 2019. Retrieved April 4, 2020, from <https://blog.f-secure.com/attack-landscape-h1-2019-iot-smb-traffic-abound/>.

[30] NUSSBAUMER, H. J. The fast Fourier transform. In *Fast Fourier Transform and Convolution Algorithms*, s. 80–111. Springer, 1981.

[31] OIKARINEN J., R. D. Internet Relay Chat Protocol. RFC 1459, 1993. Retrieved April 5, 2020, from <https://tools.ietf.org/html/rfc1459>.

[32] OIKARINEN J., R. D. Algorithms, Internet Relay Chat Protocol. RFC 1459, 1993. Retrieved April 5, 2020, from <https://tools.ietf.org/html/rfc1459>.

[33] OIKARINEN J., R. D. Architecture, Internet Relay Chat Protocol. RFC 1459, 1993. Retrieved April 5, 2020, from <https://tools.ietf.org/html/rfc1459>.

[34] OIKARINEN J., R. D. Client, Internet Relay Chat Protocol. RFC 1459, 1993. Retrieved April 5, 2020, from <https://tools.ietf.org/html/rfc1459>.

[35] OIKARINEN J., R. D. Network Congestion, Internet Relay Chat Protocol. RFC 1459, 1993. Retrieved April 5, 2020, from <https://tools.ietf.org/html/rfc1459>.

[36] OIKARINEN J., R. D. Server, Internet Relay Chat Protocol. RFC 1459, 1993. Retrieved April 5, 2020, from <https://tools.ietf.org/html/rfc1459>.

[37] P VIXIE, Y. R. S. T. Dynamic Updates In The Domain Name System (DNS Update), 1997. Retrieved June 18, 2020, from <http://www.faqs.org/rfcs/rfc2136.html>.

[38] PRENEK, O. IRC-IoT-Malware-Detection, . Retrieved August 1, 2020, from <https://github.com/stratosphereips/IRC-IoT-Malware-Detection>.

[39] PRENEK, O. Zeek: New IRC Feature Extractor Package, . Retrieved July 10, 2020, from <https://www.stratosphereips.org/blog/2020/02/13/zeek-new-irc-feature-extractor-package>.

[40] PROKOFIEV, A. O. – SMIRNOVA, Y. S. – SUROV, V. A. A method to detect Internet of Things botnets. In *2018 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*, s. 105–108. IEEE, 2018.

[41] RAMACHANDRAN, A. et al. Revealing botnet membership using DNSBL counter-intelligence. *SRUTI*. 2006, 6, s. 49–54.

[42] SAHA B., G. A. Botnet: An Overview. *CERT-In White PaperCIWP-200S-0S*. 2005.

[43] SILVA, S. S. et al. Botnets: A survey. *Computer Networks*. 2013, 57, 2, s. 378–403.

[44] STRAYER, W. T. et al. Detecting botnets with tight command and control. In *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, s. 195–202. IEEE, 2006.

[45] VALEROS, V. Timeline of IoT Malware - Version 1, 2020. Retrieved June 18, 2020, from <https://www.stratosphereips.org/blog/2020/4/26/timeline-of-iot-malware-version-1>.

[46] ZOU, C. C. – CUNNINGHAM, R. Honeypot-aware advanced botnet construction and maintenance. In *International Conference on Dependable Systems and Networks (DSN'06)*, s. 199–208. IEEE, 2006.

Appendix A

Learning Curves

A.1 IRC 3-tuple Session Detection

A.1.1 Unbalanced Dataset

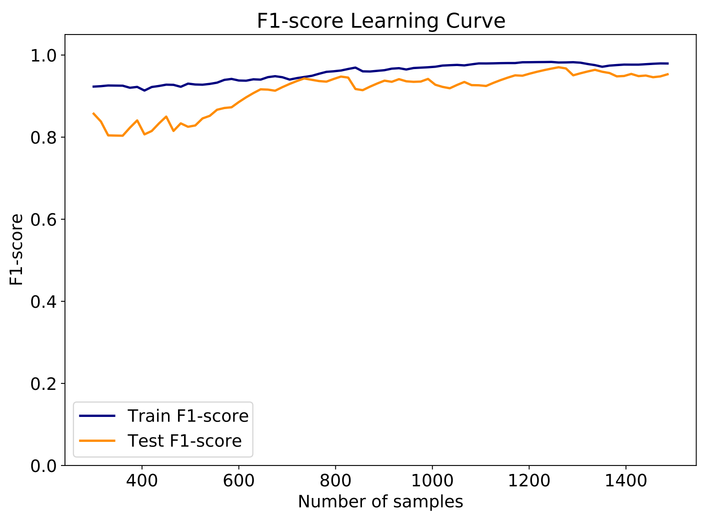

Figure A.1: IRC 3-tuple Session Detection: Learning curve of Linear Classifier model trained on unbalanced dataset.

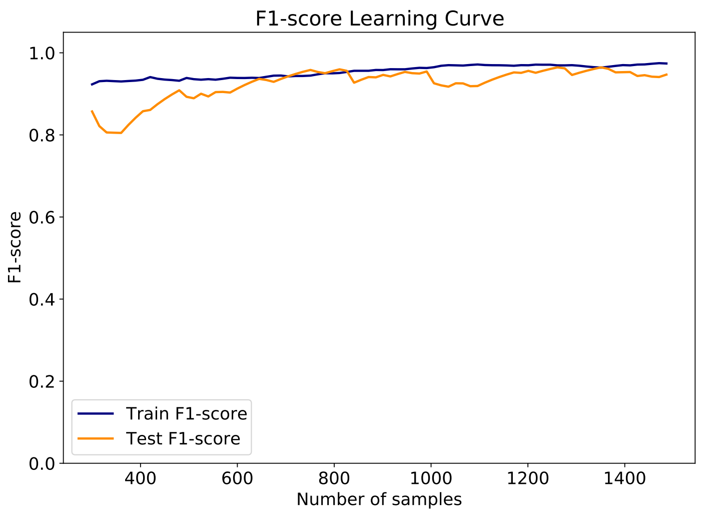

Figure A.2: IRC 3-tuple Session Detection: Learning curve of Logistic Regression model trained on unbalanced dataset.

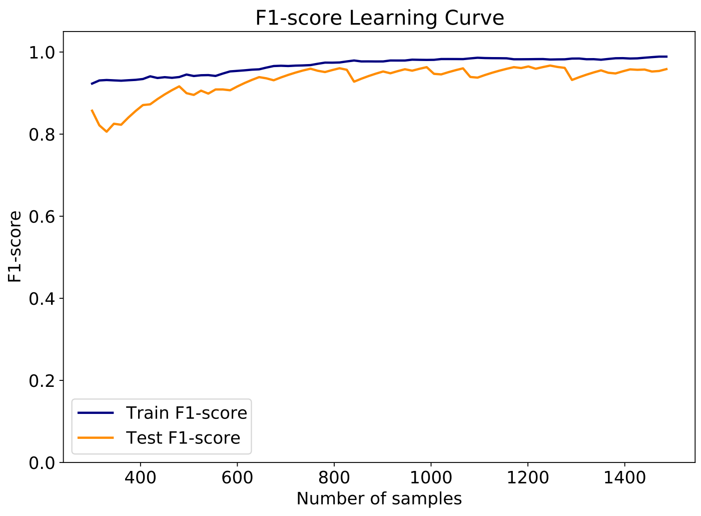

Figure A.3: IRC 3-tuple Session Detection: Learning curve of SVM model trained on unbalanced dataset.

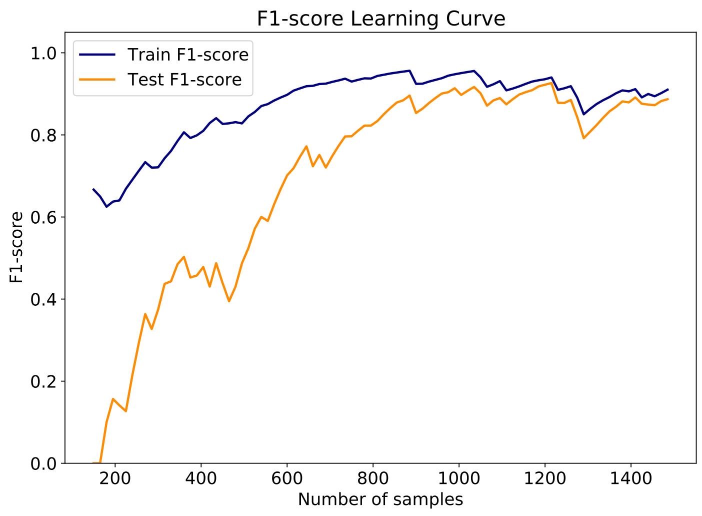

Figure A.4: IRC 3-tuple Session Detection: Learning curve of Random Forrest model trained on unbalanced dataset.

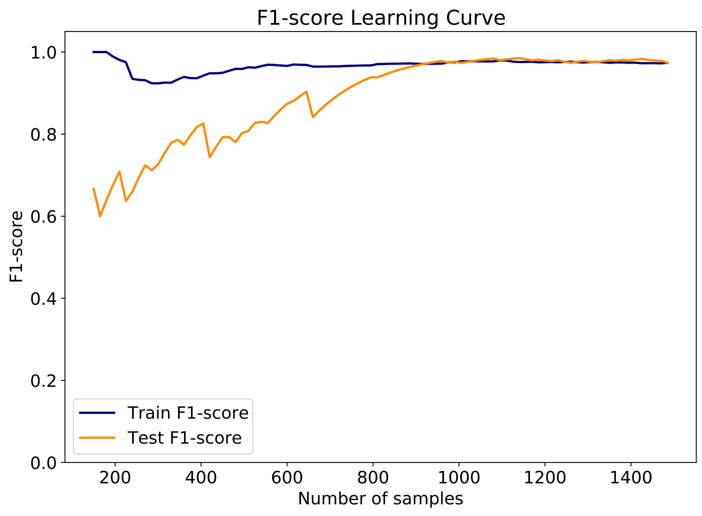

Figure A.5: IRC 3-tuple Session Detection: Learning curve of XGBoost model trained on unbalanced dataset.

A.1.2 Balanced Dataset: Undersampling

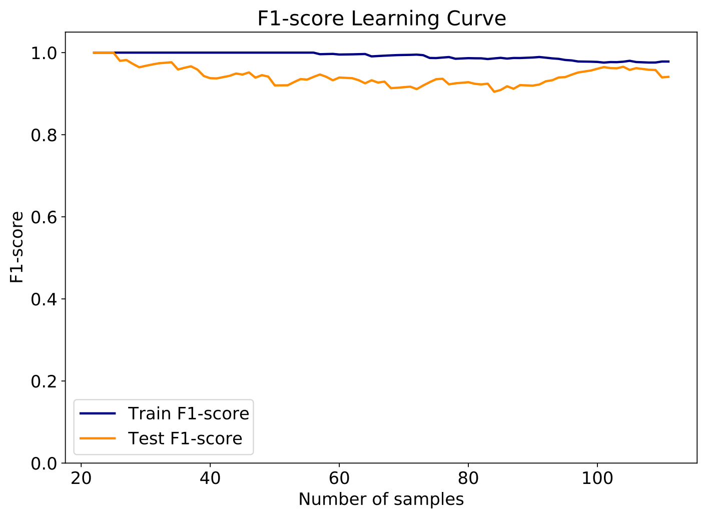

Figure A.6: IRC 3-tuple Session Detection: Learning curve of Linear Classifier model trained on undersampled dataset.

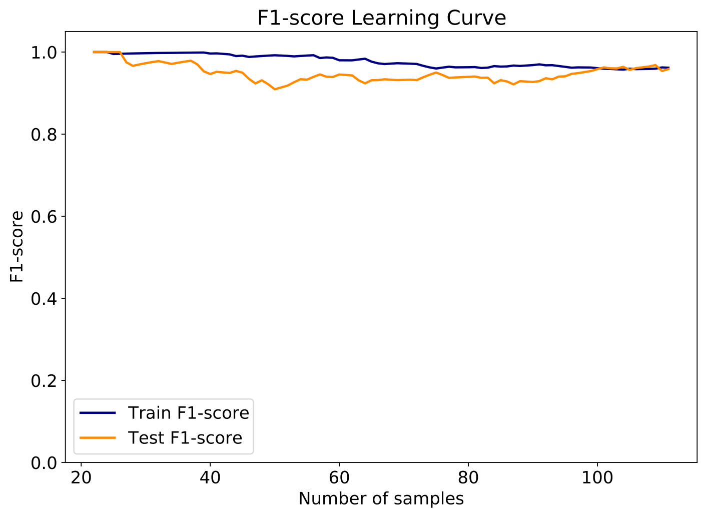

Figure A.7: IRC 3-tuple Session Detection: Learning curve of Logistic Regression model trained on undersampled dataset.

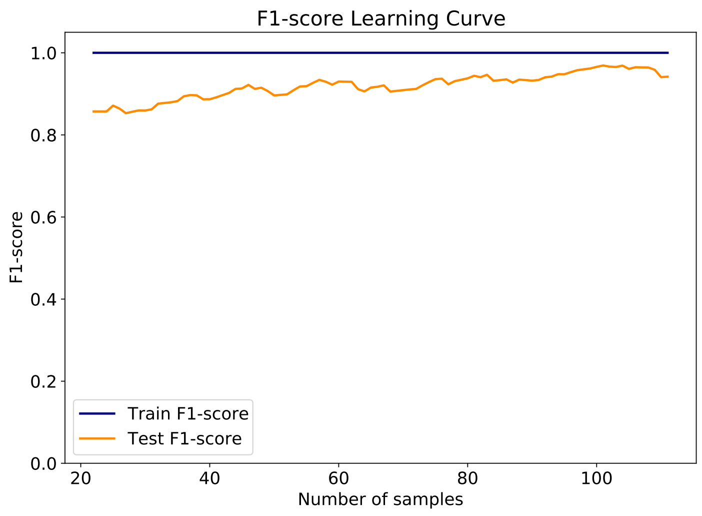

Figure A.8: IRC 3-tuple Session Detection: Learning curve of SVM model trained on undersampled dataset.

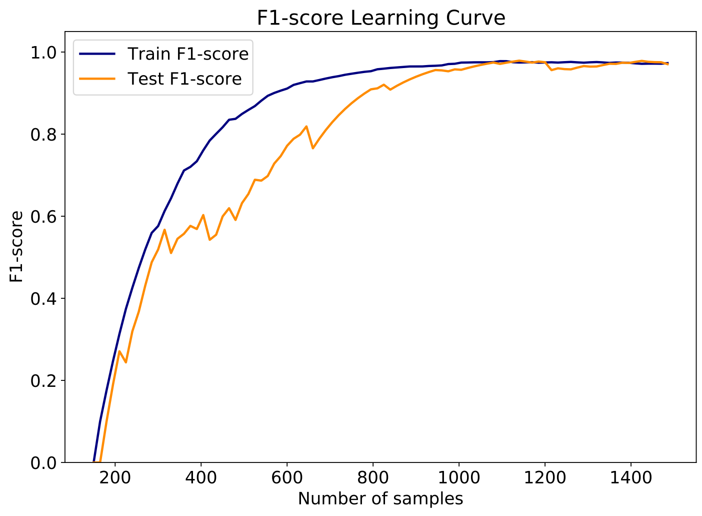

Figure A.9: IRC 3-tuple Session Detection: Learning curve of Random Forrest model trained on undersampled dataset.

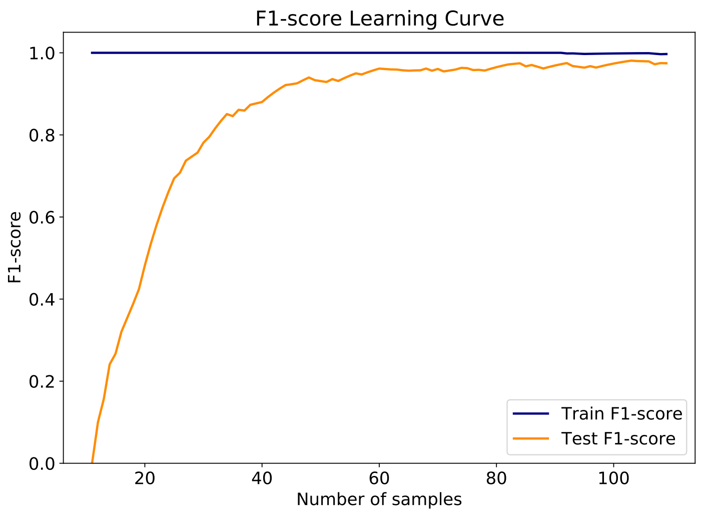

Figure A.10: IRC 3-tuple Session Detection: Learning curve of XGBoost model trained on undersampled dataset.

A.1.3 Balanced Dataset: Oversampling

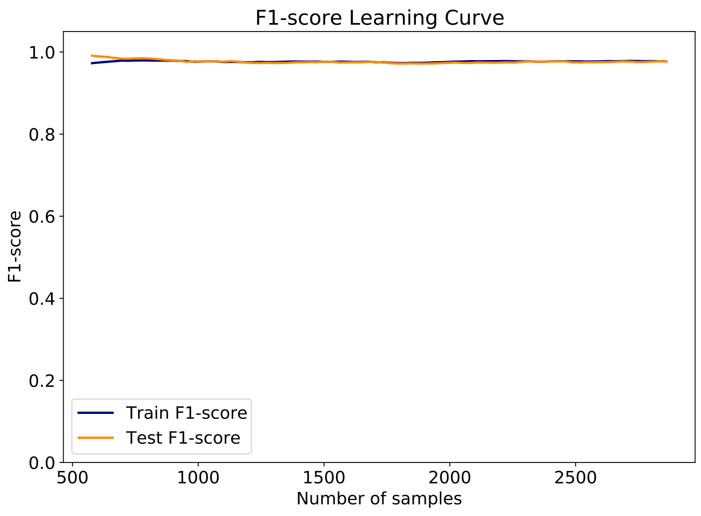

Figure A.11: IRC 3-tuple Session Detection: Learning curve of Linear Classifier model trained on oversampled dataset.

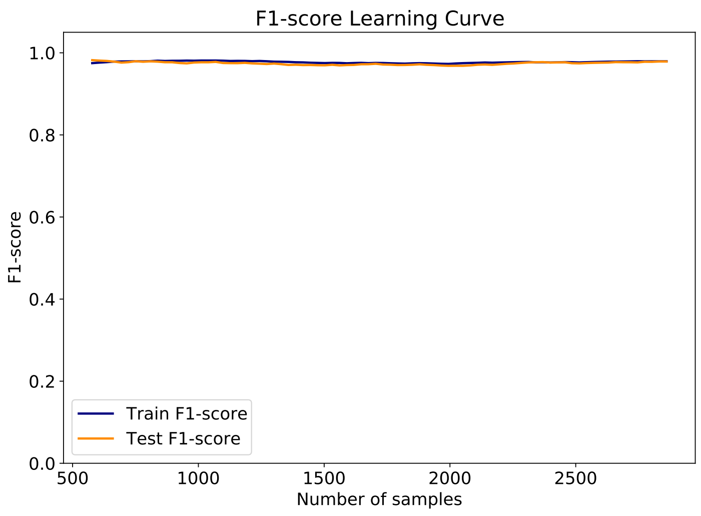

Figure A.12: IRC 3-tuple Session Detection: Learning curve of Logistic Regression model trained on oversampled dataset.

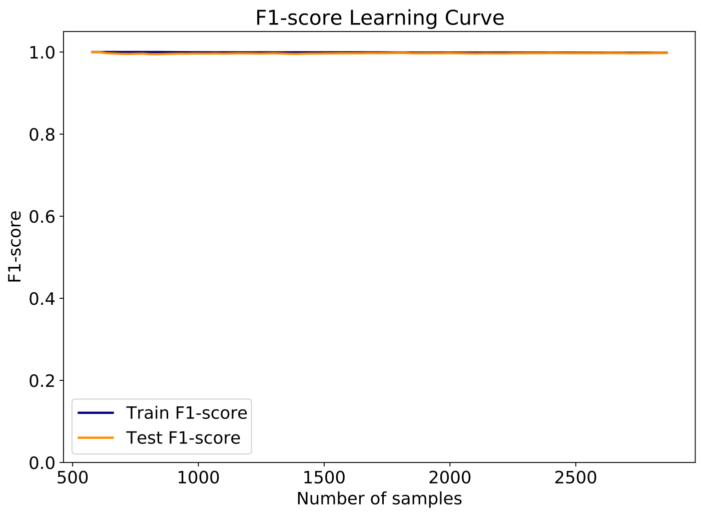

Figure A.13: IRC 3-tuple Session Detection: Learning curve of SVM model trained on oversampled dataset.

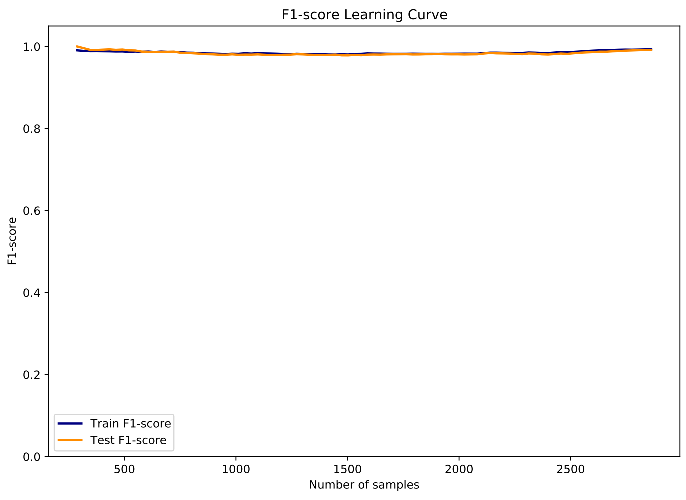

Figure A.14: IRC 3-tuple Session Detection: Learning curve of XGBoost model trained on oversampled dataset.

A.2 IRC Channel & User Detection

A.2.1 IRC Channel Detection

A.2.1.1 Unbalanced Dataset

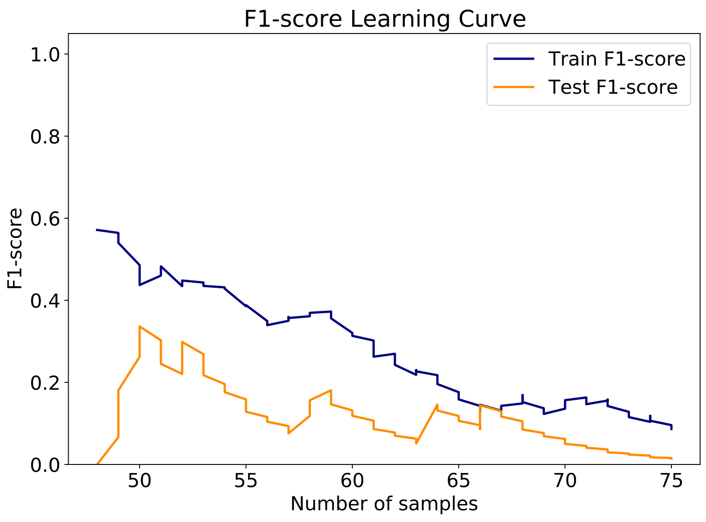

Figure A.15: IRC Channel Detection: Learning curve of Linear Classifier model trained on unbalanced dataset.

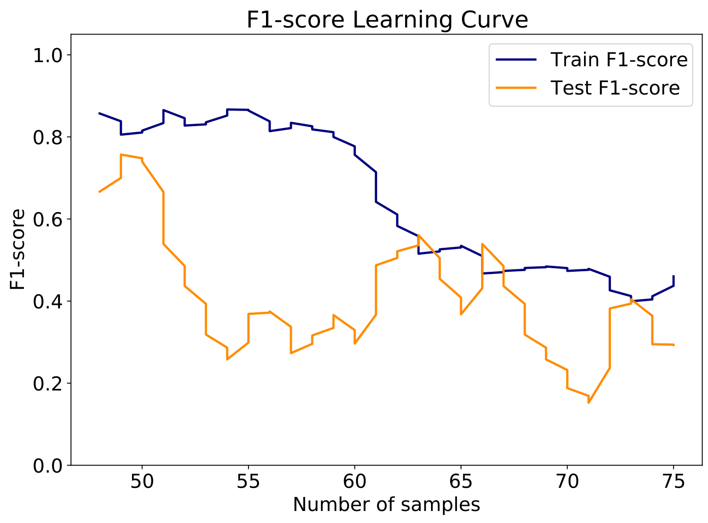

Figure A.16: IRC Channel Detection: Learning curve of Logistic Regression model trained on unbalanced dataset.

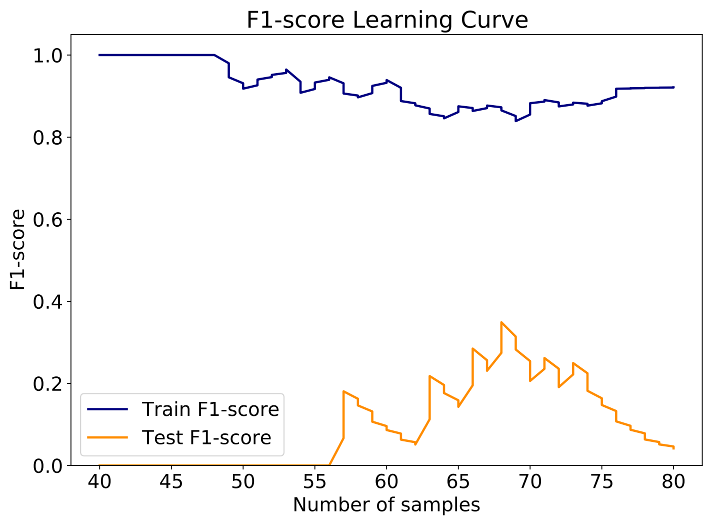

Figure A.17: IRC Channel Detection: Learning curve of SVM model trained on unbalanced dataset.

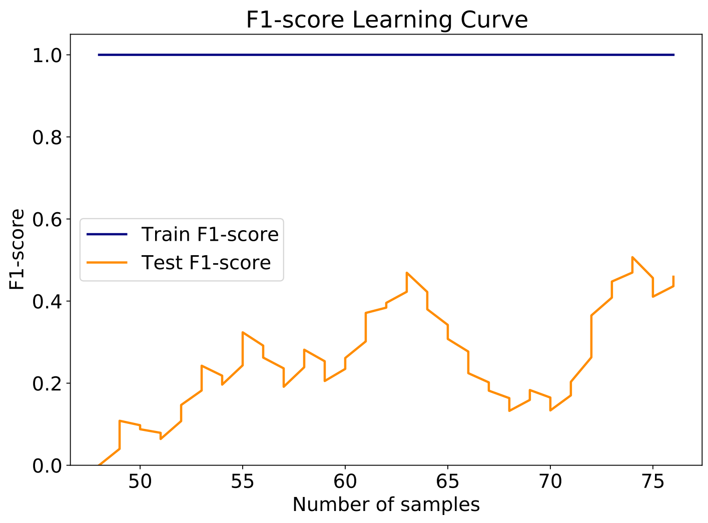

Figure A.18: IRC Channel Detection: Learning curve of XGBoost model trained on unbalanced dataset.

A.2.1.2 Balanced Dataset: Undersampling

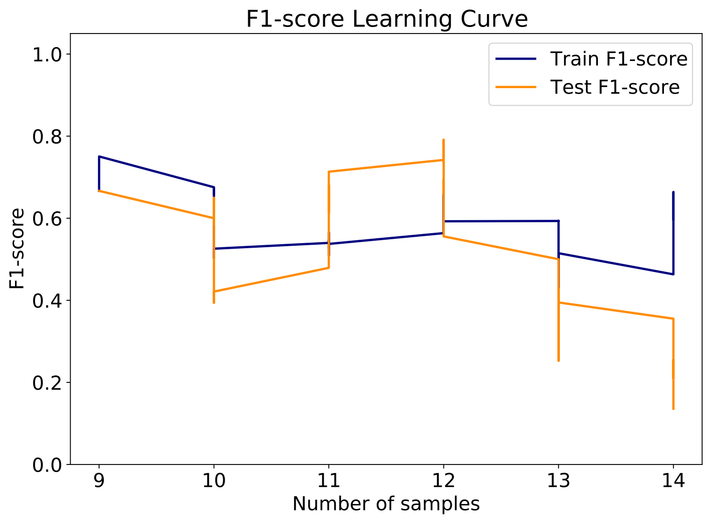

Figure A.19: IRC Channel Detection: Learning curve of Linear Classifier model trained on undersampled dataset.

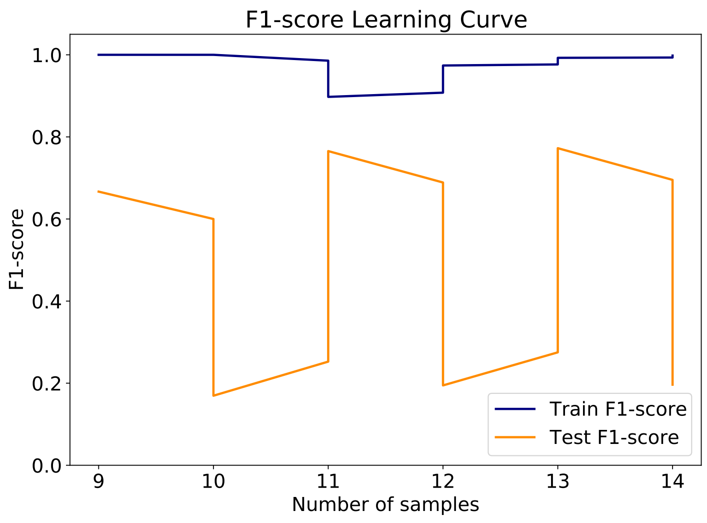

Figure A.20: IRC Channel Detection: Learning curve of Logistic Regression model trained on undersampled dataset.

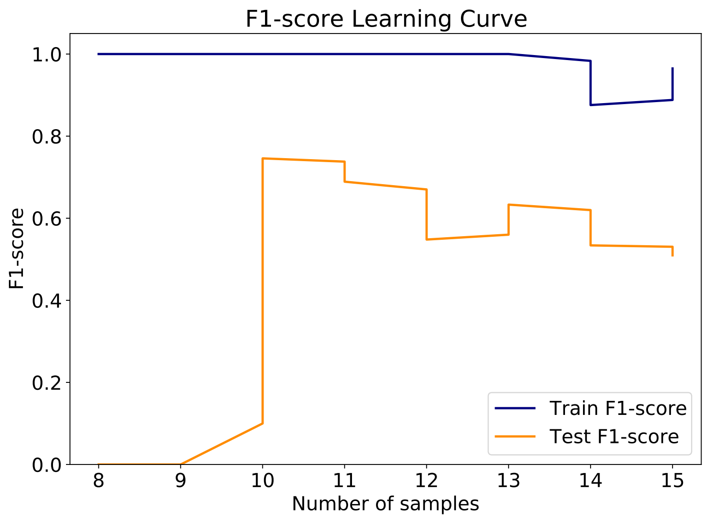

Figure A.21: IRC Channel Detection: Learning curve of SVM model trained on undersampled dataset.

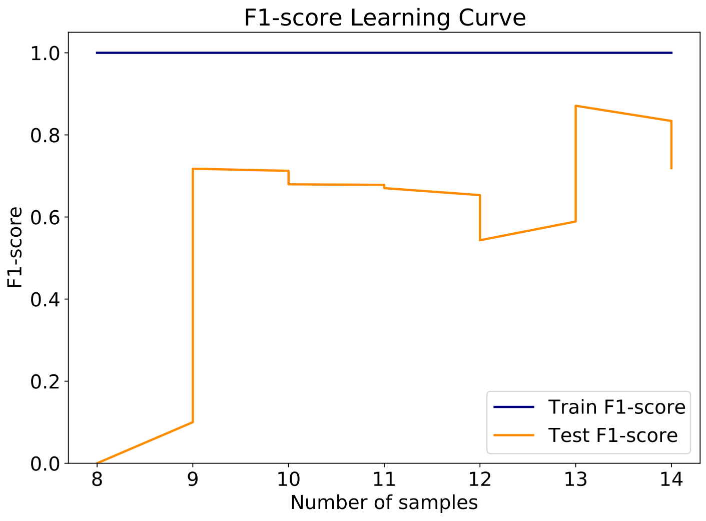

Figure A.22: IRC Channel Detection: Learning curve of XGBoost model trained on undersampled dataset.

A.2.1.3 Balanced Dataset: Oversampling

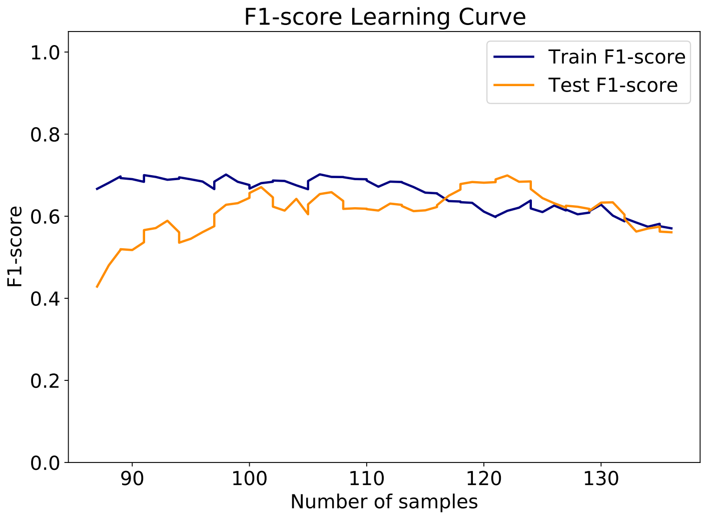

Figure A.23: IRC Channel Detection: Learning curve of Linear Classifier model trained on oversampled dataset.

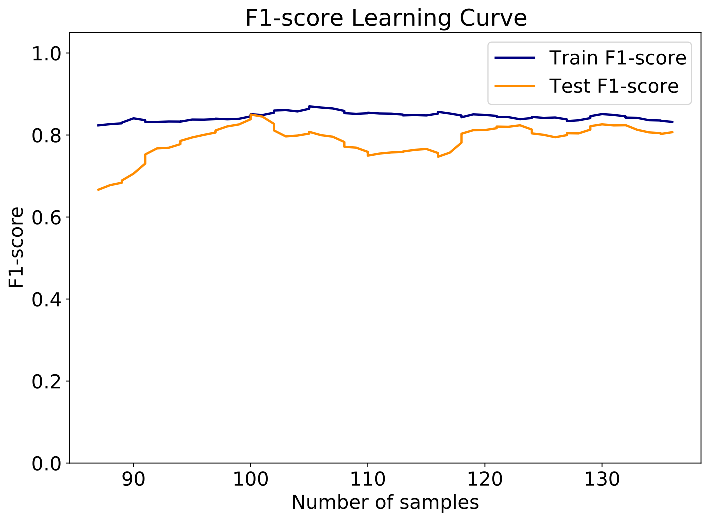

Figure A.24: IRC Channel Detection: Learning curve of Logistic Regression model trained on oversampled dataset.

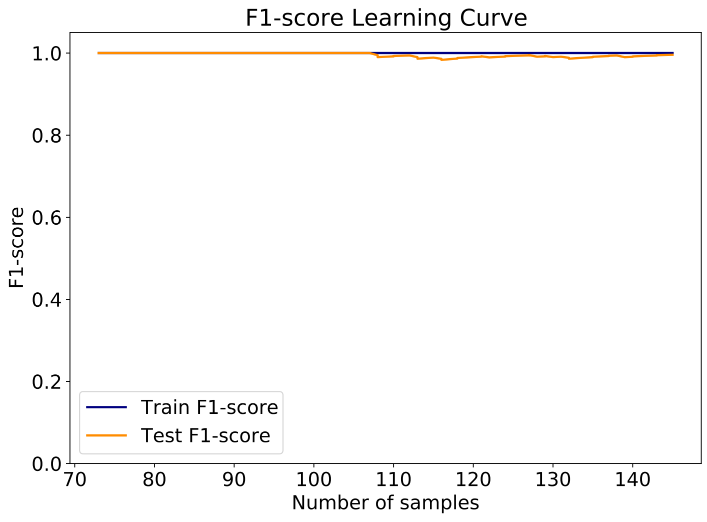

Figure A.25: IRC Channel Detection: Learning curve of SVM model trained on oversampled dataset.

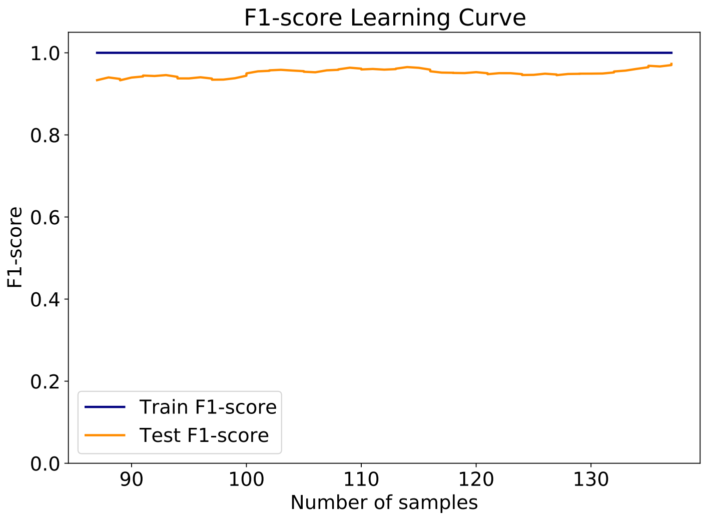

Figure A.26: IRC Channel Detection: Learning curve of XGBoost model trained on over-sampled dataset.

A.2.2 IRC User Detection

A.2.2.1 Unbalanced Dataset

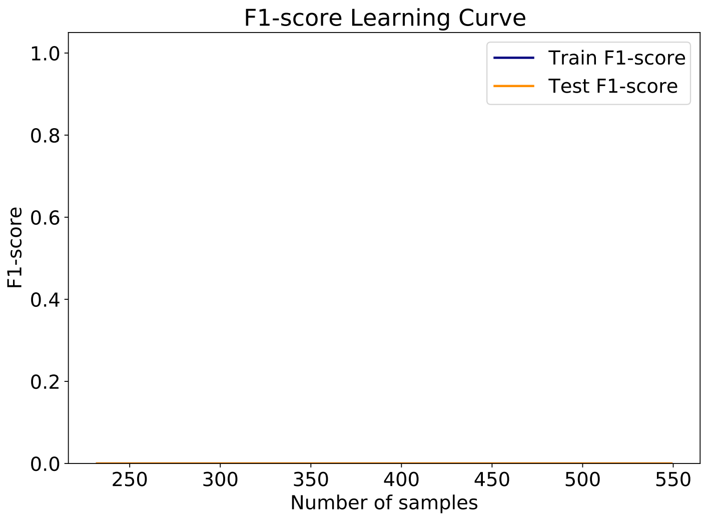

Figure A.27: IRC User Detection: Learning curve of Logistic Regression model trained on unbalanced dataset.

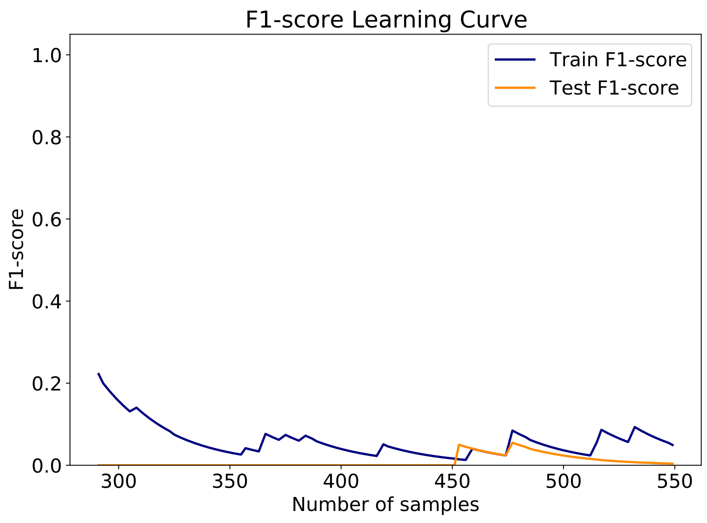

Figure A.28: IRC User Detection: Learning curve of SVM model trained on unbalanced dataset.

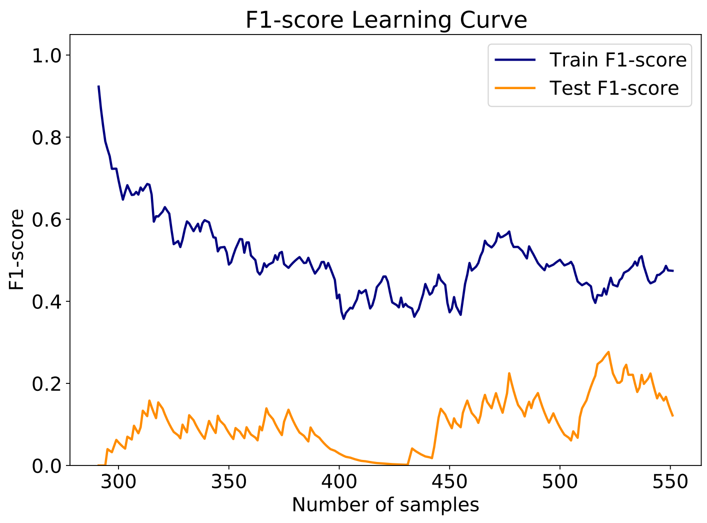

Figure A.29: IRC User Detection: Learning curve of XGBoost model trained on unbalanced dataset.

A.2.2.2 Balanced Dataset: Undersampling

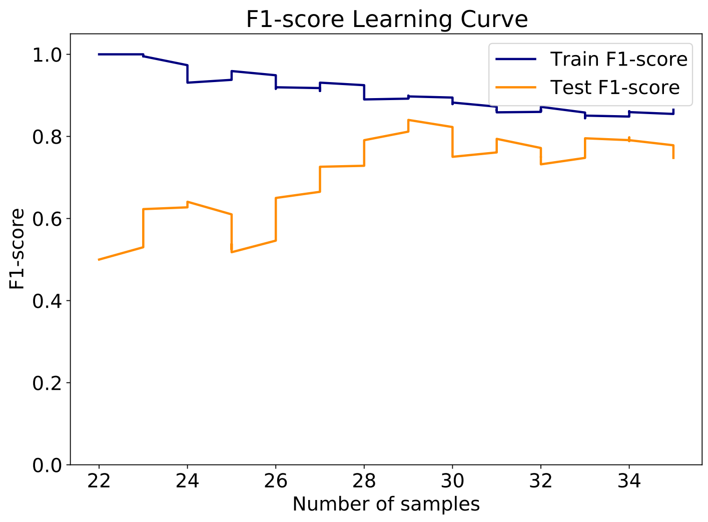

Figure A.30: IRC User Detection: Learning curve of Linear Classifier model trained on undersampled dataset.

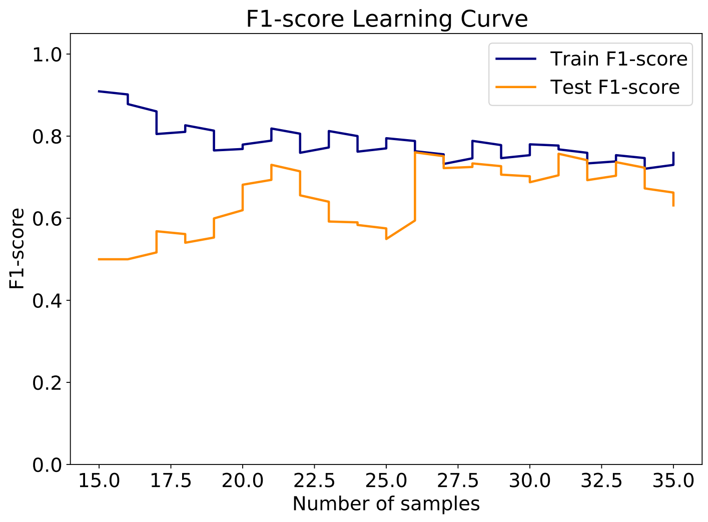

Figure A.31: IRC User Detection: Learning curve of Logistic Regression model trained on undersampled dataset.

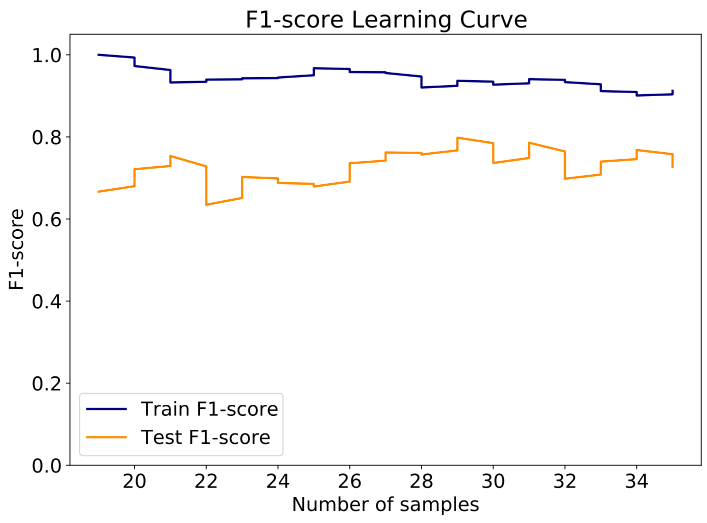

Figure A.32: IRC User Detection: Learning curve of SVM model trained on undersampled dataset.

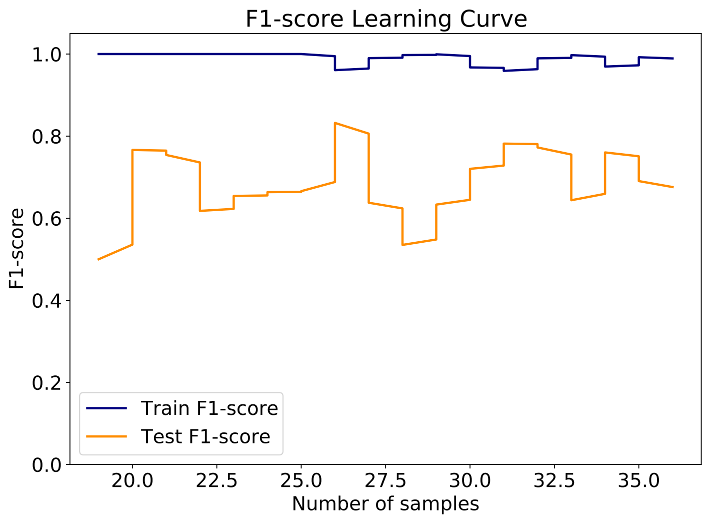

Figure A.33: IRC User Detection: Learning curve of XGBoost model trained on undersampled dataset.

A.2.2.3 Balanced Dataset: Oversampling

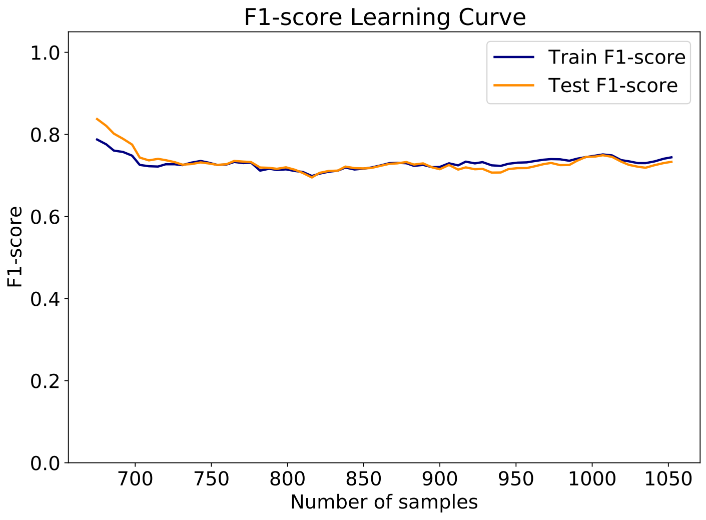

Figure A.34: IRC User Detection: Learning curve of Linear Classifier model trained on oversampled dataset.

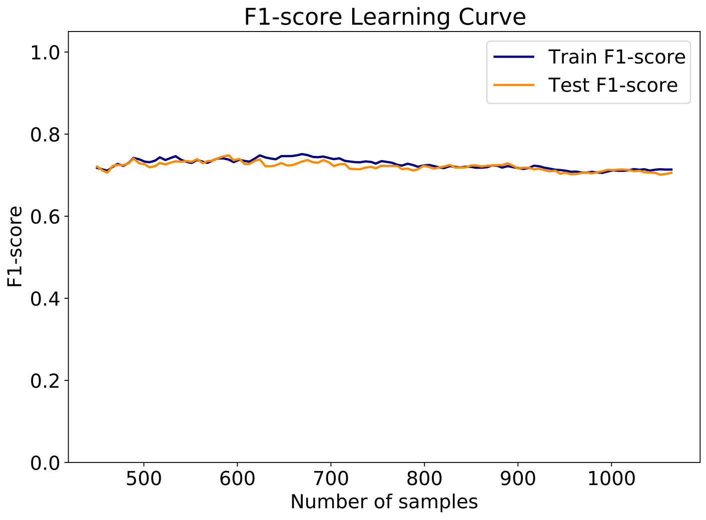

Figure A.35: IRC User Detection: Learning curve of Logistic Regression model trained on oversampled dataset.

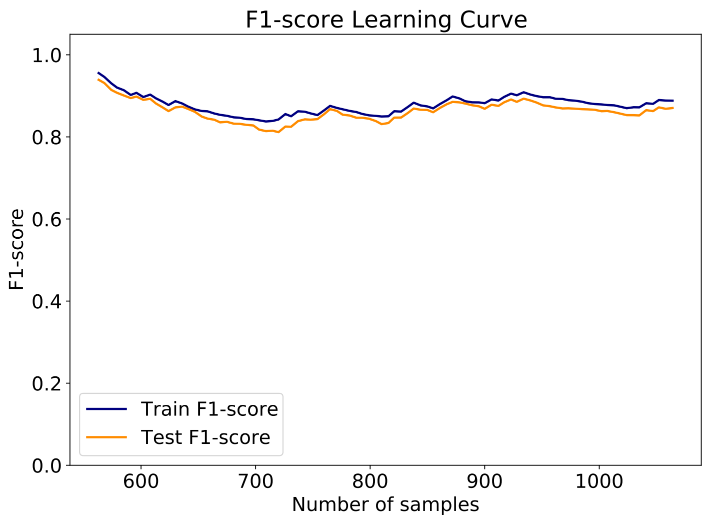

Figure A.36: IRC User Detection: Learning curve of SVM model trained on oversampled dataset.

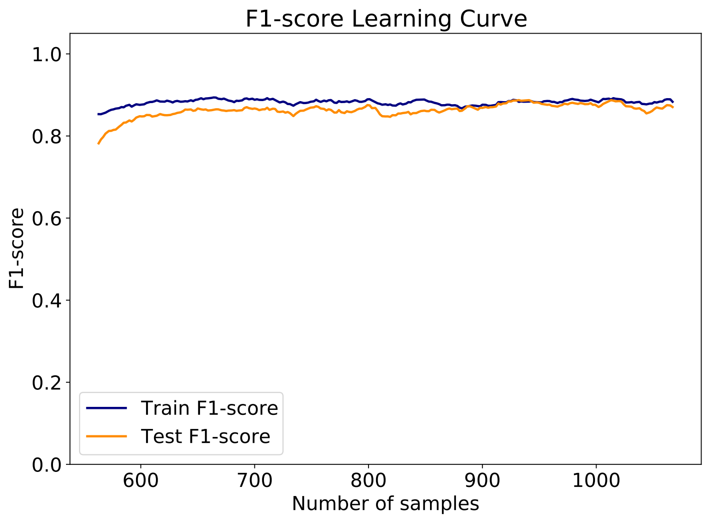

Figure A.37: IRC User Detection: Learning curve of XGBoost model trained on oversampled dataset.

Appendix B

IDS Analysis Outputs

B.1 Snort

```
07/24-07:15:10.367164  [**] [1:52824:1] MALWARE-CNC Unix.Trojan.Muhstik
↪  variant IRC outbound connection [**] [Classification: A Network Trojan
↪  was detected] [Priority: 1] {TCP} 192.168.100.103:35990 ->
↪  111.230.241.23:2407
07/24-09:53:22.513899  [**] [1:52824:1] MALWARE-CNC Unix.Trojan.Muhstik
↪  variant IRC outbound connection [**] [Classification: A Network Trojan
↪  was detected] [Priority: 1] {TCP} 192.168.100.103:40328 ->
↪  185.61.149.22:2407
08/13-14:10:29.973233  [**] [1:52824:1] MALWARE-CNC Unix.Trojan.Muhstik
↪  variant IRC outbound connection [**] [Classification: A Network Trojan
↪  was detected] [Priority: 1] {TCP} 192.168.100.103:55770 ->
↪  185.61.149.22:2407
08/20-18:54:39.114303  [**] [1:52824:1] MALWARE-CNC Unix.Trojan.Muhstik
↪  variant IRC outbound connection [**] [Classification: A Network Trojan
↪  was detected] [Priority: 1] {TCP} 192.168.100.103:37472 ->
↪  111.230.241.23:2407
08/23-05:52:00.151092  [**] [1:52824:1] MALWARE-CNC Unix.Trojan.Muhstik
↪  variant IRC outbound connection [**] [Classification: A Network Trojan
↪  was detected] [Priority: 1] {TCP} 192.168.100.103:40538 ->
↪  185.61.149.22:2407
07/22-04:15:10.227886  [**] [1:52824:1] MALWARE-CNC Unix.Trojan.Muhstik
↪  variant IRC outbound connection [**] [Classification: A Network Trojan
↪  was detected] [Priority: 1] {TCP} 192.168.100.103:49076 ->
↪  185.61.149.22:2407
08/23-23:49:48.304413  [**] [1:52824:1] MALWARE-CNC Unix.Trojan.Muhstik
↪  variant IRC outbound connection [**] [Classification: A Network Trojan
↪  was detected] [Priority: 1] {TCP} 192.168.100.103:56586 ->
↪  185.61.149.22:2407
```

```
08/20-19:35:13.755848  [**] [1:52824:1] MALWARE-CNC Unix.Trojan.Muhstik
 ↪  variant IRC outbound connection [**] [Classification: A Network Trojan
 ↪  was detected] [Priority: 1] {TCP} 192.168.100.103:39922 ->
 ↪  111.230.241.23:2407
07/21-17:15:50.475528  [**] [1:52824:1] MALWARE-CNC Unix.Trojan.Muhstik
 ↪  variant IRC outbound connection [**] [Classification: A Network Trojan
 ↪  was detected] [Priority: 1] {TCP} 192.168.100.103:40964 ->
 ↪  185.61.149.22:2407
07/14-07:44:00.347845  [**] [1:52824:1] MALWARE-CNC Unix.Trojan.Muhstik
 ↪  variant IRC outbound connection [**] [Classification: A Network Trojan
 ↪  was detected] [Priority: 1] {TCP} 192.168.100.103:50444 ->
 ↪  185.61.149.22:2407
07/25-05:15:36.894465  [**] [1:52824:1] MALWARE-CNC Unix.Trojan.Muhstik
 ↪  variant IRC outbound connection [**] [Classification: A Network Trojan
 ↪  was detected] [Priority: 1] {TCP} 192.168.100.103:56626 ->
 ↪  185.61.149.22:2407
07/22-02:53:48.007985  [**] [1:52824:1] MALWARE-CNC Unix.Trojan.Muhstik
 ↪  variant IRC outbound connection [**] [Classification: A Network Trojan
 ↪  was detected] [Priority: 1] {TCP} 192.168.100.103:44728 ->
 ↪  111.230.241.23:2407
07/04-22:35:11.246265  [**] [1:52824:1] MALWARE-CNC Unix.Trojan.Muhstik
 ↪  variant IRC outbound connection [**] [Classification: A Network Trojan
 ↪  was detected] [Priority: 1] {TCP} 192.168.100.103:51054 ->
 ↪  185.61.149.22:2407
```

B.2 Suricata

```
05/21/2019-02:08:18.952425  [**] [1:2000345:16] ET MALWARE IRC Nick change on
 ↪  non-standard port [**] [Classification: A Network Trojan was detected]
 ↪  [Priority: 1] {TCP} 192.168.1.196:43736 -> 149.56.41.138:80
05/21/2019-02:08:18.952430  [**] [1:2000345:16] ET MALWARE IRC Nick change on
 ↪  non-standard port [**] [Classification: A Network Trojan was detected]
 ↪  [Priority: 1] {TCP} 192.168.1.196:43736 -> 149.56.41.138:80
05/21/2019-02:08:21.056473  [**] [1:2000348:15] ET POLICY IRC Channel JOIN on
 ↪  non-standard port [**] [Classification: A Network Trojan was detected]
 ↪  [Priority: 1] {TCP} 192.168.1.196:43736 -> 149.56.41.138:80
05/21/2019-02:08:21.056478  [**] [1:2000348:15] ET POLICY IRC Channel JOIN on
 ↪  non-standard port [**] [Classification: A Network Trojan was detected]
 ↪  [Priority: 1] {TCP} 192.168.1.196:43736 -> 149.56.41.138:80
05/21/2019-03:20:31.646506  [**] [1:2000355:5] ET CHAT IRC authorization
 ↪  message [**] [Classification: Misc activity] [Priority: 3] {TCP}
 ↪  149.56.41.138:80 -> 192.168.1.196:43736
05/21/2019-03:23:26.124409  [**] [1:2000347:16] ET MALWARE IRC Private
 ↪  message on non-standard port [**] [Classification: A Network Trojan was
 ↪  detected] [Priority: 1] {TCP} 192.168.1.196:43736 -> 149.56.41.138:80
```

```
05/21/2019-03:23:26.124422  [**] [1:2000347:16] ET MALWARE IRC Private
↪  message on non-standard port [**] [Classification: A Network Trojan was
↪  detected] [Priority: 1] {TCP} 192.168.1.196:43736 -> 149.56.41.138:80
05/21/2019-03:23:38.303203  [**] [1:2000347:16] ET MALWARE IRC Private
↪  message on non-standard port [**] [Classification: A Network Trojan was
↪  detected] [Priority: 1] {TCP} 192.168.1.196:43736 -> 149.56.41.138:80
05/21/2019-03:23:38.303217  [**] [1:2000347:16] ET MALWARE IRC Private
↪  message on non-standard port [**] [Classification: A Network Trojan was
↪  detected] [Priority: 1] {TCP} 192.168.1.196:43736 -> 149.56.41.138:80
05/21/2019-03:29:03.583108  [**] [1:2000347:16] ET MALWARE IRC Private
↪  message on non-standard port [**] [Classification: A Network Trojan was
↪  detected] [Priority: 1] {TCP} 192.168.1.196:43736 -> 149.56.41.138:80
05/21/2019-03:29:03.583121  [**] [1:2000347:16] ET MALWARE IRC Private
↪  message on non-standard port [**] [Classification: A Network Trojan was
↪  detected] [Priority: 1] {TCP} 192.168.1.196:43736 -> 149.56.41.138:80
05/21/2019-03:29:09.461491  [**] [1:2000347:16] ET MALWARE IRC Private
↪  message on non-standard port [**] [Classification: A Network Trojan was
↪  detected] [Priority: 1] {TCP} 192.168.1.196:43736 -> 149.56.41.138:80
05/21/2019-09:18:20.276178  [**] [1:2000347:16] ET MALWARE IRC Private
↪  message on non-standard port [**] [Classification: A Network Trojan was
↪  detected] [Priority: 1] {TCP} 192.168.1.196:43736 -> 149.56.41.138:80
05/21/2019-02:08:18.850733  [**] [1:2210026:2] SURICATA STREAM ESTABLISHED
↪  SYN resend [**] [Classification: Generic Protocol Command Decode]
↪  [Priority: 3] {TCP} 192.168.1.196:43736 -> 149.56.41.138:80
05/21/2019-02:08:18.850738  [**] [1:2210026:2] SURICATA STREAM ESTABLISHED
↪  SYN resend [**] [Classification: Generic Protocol Command Decode]
↪  [Priority: 3] {TCP} 192.168.1.196:43736 -> 149.56.41.138:80
05/21/2019-02:08:18.951676  [**] [1:2210023:2] SURICATA STREAM ESTABLISHED
↪  SYNACK resend with different ACK [**] [Classification: Generic Protocol
↪  Command Decode] [Priority: 3] {TCP} 149.56.41.138:80 ->
↪  192.168.1.196:43736
05/21/2019-02:08:18.951681  [**] [1:2210023:2] SURICATA STREAM ESTABLISHED
↪  SYNACK resend with different ACK [**] [Classification: Generic Protocol
↪  Command Decode] [Priority: 3] {TCP} 149.56.41.138:80 ->
↪  192.168.1.196:43736
05/21/2019-02:08:18.951924  [**] [1:2210044:2] SURICATA STREAM Packet with
↪  invalid timestamp [**] [Classification: Generic Protocol Command Decode]
```