ISTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

AN OPEN-SOURCE, MACHINE LEARNING BASED INTRUSION DETECTION SYSTEM

M.Sc. THESIS

Zemre ARSLAN TÜVER

Department of Computational Science and Engineering

Computational Science and Engineering Programme

SEPTEMBER 2019

ISTANBUL TECHNICAL UNIVERSITY ★ INFORMATICS INSTITUTE

AN OPEN-SOURCE, MACHINE LEARNING BASED INTRUSION DETECTION SYSTEM

M.Sc. THESIS

Zemre ARSLAN TÜVER
(702131021)

Department of Computational Science and Engineering

Computational Science and Engineering Programme

Thesis Advisor: Assoc. Prof. Dr. Enver Özdemir

SEPTEMBER 2019

ISTANBUL TEKNİK ÜNİVERSİTESİ ★ BİLİŞİM ENSTİTÜSÜ

MAKİNA ÖĞRENMESİ TABANLI AÇIK KAYNAK KODLU SALDIRI TESPİT SİSTEMİ

YÜKSEK LİSANS TEZİ

Zemre ARSLAN TÜVER
(702131021)

Hesaplamalı Bilim ve Mühendislik Anabilim Dalı

Hesaplamalı Bilim ve Mühendislik Programı

Tez Danışmanı: Doç. Dr. Enver Özdemir

EYLÜL 2019

Zemre ARSLAN TÜVER, a M.Sc. student of ITU Informatics Institute 702131021 successfully defended the thesis entitled "AN OPEN-SOURCE, MACHINE LEARNING BASED INTRUSION DETECTION SYSTEM", which he/she prepared after fulfilling the requirements specified in the associated legislations, before the jury whose signatures are below.

Thesis Advisor :	**Assoc. Prof. Dr. Enver Özdemir** Istanbul Technical University	.....
Jury Members :	**Assoc. Prof. Dr. Behçet Uğur TÖREYİN** Istanbul Technical University	.....
	Asst. Prof. Dr. Elif Segah ÖZTAŞ Karamanoglu Mehmetbey University	.....

Date of Submission : 31 August 2019
Date of Defense : 12 September 2019

To my family

FOREWORD

I would like to express my sincere gratitude to my supervisor Assoc. Prof. Dr. Enver Özdemir for his guidance and patient understanding. It was not an easy work for both of us, since we were working from different countries. His continuous motivational support, positive attitude and his enlightening ideas helped me a lot to finish my work. I also would like to thank to my spouse Cem, and my mother for their moral support and love.

September 2019

Zemre ARSLAN TÜVER
(Computer Engineer)

TABLE OF CONTENTS

Page

FOREWORD ..... **ix**
TABLE OF CONTENTS ..... **xi**
ABBREVIATIONS ..... **xiii**
SYMBOLS ..... **xv**
LIST OF TABLES ..... **xvii**
LIST OF FIGURES ..... **xix**
SUMMARY ..... **xxi**
ÖZET ..... **xxv**
1. INTRODUCTION **1**
1.1 Purpose of Thesis 1
1.2 Background 1
1.2.1 DDoS attacks 1
1.2.2 Intrusion detection systems (IDS) 3
1.3 Literature Review 5
2. SYSTEM ARCHITECTURE **9**
2.1 The Raw Network Data 10
2.2 Audit Trail Topic 10
2.3 Data Preprocessing Module 12
2.4 Processed Data Topic 13
2.5 ML Engine 14
2.5.1 Feature selection 14
2.5.2 Label encoding 15
2.5.3 Preparing training and testing sets 15
2.5.4 Apply machine learning algorithms 17
2.5.4.1 Isolation forest 17
2.5.4.2 Elliptic envelope 18
2.5.4.3 Local outlier factor 19
2.6 Intrusion Classifier 21
2.7 Malicious Data Topic 21
2.8 Elastic Search & Kibana 22
3. PROOF OF CONCEPT STUDIES **23**
3.1 The Anatomy of the Dataset 23
3.2 Our Approach for Botnet Detection 24
3.3 Data Preparation and Transformation 26
3.3.1 Data manipulation 26
3.3.2 Feature selection 29
3.3.3 Label encoding 29

3.4 Preparing Train and Test Data 30
4. RESULTS 33
4.1 Number of Different Protocol Types 34
4.2 Average Interarrival Time 34
4.3 Average Packet Length 35
4.4 Same-length number of packets ratio 36
5. OPEN ISSUES & CHALLENGES & FUTURE WORK 37
6. CONCLUSIONS 39
REFERENCES 41
CURRICULUM VITAE 45

ABBREVIATIONS

ACL : Access Control List
API : Application Programming Interface
C&C : Command and Control Server
CIDS : Common Intrusion Detection Framework
CTU-13 : Czech Technical University's 13 Different Malware Dataset
DARPA : The Defense Advanced Research Projects Agency
DDoS : Distributed Denial of Service
DNS : Domain Name System
DSLAM : Digital Subscriber Line Access Multiplexer
dTos : Destination Type of Service Byte Value
EE : Elliptic Envelope
ELK Stack : Elastic Stack
FMCD : FAST-Minimum Covariance Determinate
FP : False Positives
HTTP : Hyper-Text Transfer Protocol
HTTPS : Secure Hyper-Text Transfer Protocol
HIDS : Host-based Intrusion Detection System
IC : Intrusion Classifier
ICMP : Internet Control Message Protocol
IDS : Intrusion Detection System
IP : Internet Protocol
ISOF : Isolation Forest
JSON : JavaScript Object Notation
kNN : k-Nearest Neighbors
LOF : Local Outlier Factor
MD5 : Message-Digest Algorithm 5
ML : Machine Learning
NFIDS : Neuro Fizzy Intrusion Detection System
NIDS : Network-based Intrusion Detection System
OSEMN : Obtain, Scrub, Explore, Model, Interpret
POC : Proof of Concept
REST : Representational State Transfer
RIC : Regional Infection Coefficient
RNG : Random Number Generator
SMTP : Simple Mail Transfer Protocol
sTos : Source Type of Service Byte Value
TP : True Positive
UDP : Universal Datagram Protocol

SYMBOLS

λ	: Average inter-arrival time of packets from a specific host
D	: Average data rate
d_{mh}	: Mahalanobis distance
L	: Average packet length
N_p	: Number of incoming packets which belongs to one specific host IP
r	: Recall
R_i	: Infected region
R_b	: Benign region
S_p	: Total size in bits arrived in a time window
T_w	: Size of time window in seconds

LIST OF TABLES

Page

Table 2.1 : List of fields in Audit trails of proposed framework. **12**
Table 3.1 : Scenario 10 characteristics of CTU-13 dataset. **24**
Table 3.2 : List of infected hosts with their assigned regions. **27**
Table 3.3 : List of CTU-13 Scenario 10 dataset columns. **23**
Table 4.1 : Outlier detection results with default features. **33**
Table 4.2 : Outlier detection results with protocol type count. **34**
Table 4.3 : Outlier detection results with interarival time..... **35**
Table 4.4 : Outlier detection results with same-length packet ratio. **35**

LIST OF FIGURES

Page

Figure 1.1 : Layered architecture of a DDoS attack. **2**
Figure 1.2 : Illustration of a Common Intrusion Detection Framework. **3**
Figure 2.1 : System model of proposed IDS. **9**
Figure 2.2 : Relationship between audit trail topic and consumers. **11**
Figure 2.3 : Fields of flow identifier **13**
Figure 2.4 : ML ready data fields after preprocessing. **13**
Figure 2.5 : Ratio of train:test data used in our system model. **16**
Figure 2.6 : Split train and test data using Scikit-learn. **16**
Figure 2.7 : Graphical representation of isolation forest [1] [2]. Retrieved from scikit-learn 0.21.3 documentation. **18**
Figure 2.8 : Application of Isolation Forest algorithm in scikit-learn library. **18**
Figure 2.9 : Mahalanobis distances of a dataset after the application of Elliptic Envelope routine [1] [2]. Retrieved from scikit-learn 0.21.3 documentation. ... **19**
Figure 2.10 : Application of Elliptic Envelope algorithm in scikit-learn library. **19**
Figure 2.11 : Distance based outlier detection approach. **20**
Figure 2.12 : Application of Local Outlier Factor algorithm in scikit-learn library. **21**
Figure 3.1 : Percentages of missing values in CTU-13 dataset. **28**
Figure 3.2 : Filling NaN rows and dropping unused columns with pandas. **28**
Figure 3.3 : Transformation and inverse transformation with label encoding. **30**
Figure 3.4 : Contamination ratios of testing and training sets. **31**
Figure 4.1 : Distribution of benign packet sizes on time domain. **35**
Figure 4.2 : Distribution of malicious packet sizes on time domain. **36**

AN OPEN-SOURCE, MACHINE LEARNING BASED INTRUSION DETECTION SYSTEM

SUMMARY

The exponential growth trend [3] in global data demand, leaded the network operators and system administrators to set up more complex infrastructures to be able to satisfy ever-changing data requirements. As a consequence, securing a complex network system or to be able to act promptly during an attack become a very troublesome business.

While network administrators are becoming more provident against intruders, intruders are also changing their methodology, and they manage to find new ways to attack to the target systems. One of the most well-known attack types is Denial-of-Service (DoS) attacks, which is done by exploiting the vulnerabilities of the target system by compromising some slave computers or botnets.

Denial-of-Service (DoS) attacks, aim to make the target system resources unavailable to its legitimate users. The DoS attack is generated by the source which is also known as Command and Control (C&C) server and C&C servers controls and uses the slave/zombie computers to consume the resources of the target system. A type of DoS attack is Distributed Denial of Service Attacks (DDoS), which the attackers use multiple botnets from various locations to exhaust the system with loaded traffic. The damage might be irrevocable for most of the companies, which do not have any DoS attack protection or strategy. The financial consequences of this kind of an attack might be large scale, since it might lead to data theft and eventually the loss of user's trust.

Thinking about the unexpected costs and consequences of an attack, it would be truly helpful, if a possible future attack is detected/predicted in an earlier phase. However, it is not easy to detect the true source during an attack, even nearly impossible, since the IP (Internet Protocol) addresses of the attacker's devices are perfectly disguised as legitimate sources, or their IP's might be spoofed.

Obviously, there are some preventive actions that are suggested for malicious attacks, such as deploying firewalls or increasing the bandwidth capacity that the system can handle. In addition to these precautions, as stated in [4], having a good understanding of the trend of the usual traffic might be more helpful to avoid possible attacks. This could be done by continuously monitoring the incoming and outgoing traffic of the system and knowing which traffic can be classified as normal and which of them can be classified as abnormal. It is also proposed in [4] that, if it is possible to deduct a base trend for the usual traffic behavior in a system, then it would be possible to reject illegitimate traffic beforehand. Although, it is a robust way to avoid unwanted traffic in a network at an earlier stage, it should not be forgotten that a system can always be a victim of DDoS attack, so we can always provide more intelligent preventive solutions with custom intrusion detection systems.

As mentioned before, it is nearly impossible to find the true attack sources under the heavy load during a distributed attack since the information of the slave computers are spoofed. Therefore, it is not easy to specify the sources and cease the network between the attacker and the victim system. However, if the locations of the attacker can be guessed, then the administrators can cease the connection in between, and thus decrease the load and look for alternative solutions not to exhaust their system in the meanwhile.

In this work, we propose an intelligent, Machine Learning (ML) based intrusion detection system, which we believe will give the network administrators particular clues about where the botnets might actually be located during an attack. We will model our system as a live monitoring system which collects the traffic data and learns from this collected traffic data. It will mark and score some hosts from incoming traffic as abnormal and thus, it will give the administrators a general insight of what is going on in the present situation and what they might be expecting for the future.

Our intrusion detection system is a novel, open-source, scalable and distributed in order to handle huge volumes of data. We designed the system as customizable and scalable, so that it can be easily set up in front of any network system to cope with intrusion attacks. The system will collect raw network data in Apache Kafka topics in specific time-windows, consumers will process this raw data in chunks and send the processed data to Machine Learning Engine. Machine Learning Engine will apply unsupervised learning algorithms on the incoming data and write the results into related topics in Kafka again. The Intrusion Classifier, which we also introduce as a Kafka consumer, will decide if the suspicious data is actually malicious or not. And after the decision of IC, the results will be written in Elastic Search for the use of system administrators. Kibana interfaces will be used to serve and visualize the results.

In our scenario, we have N regions which are allowed to send traffic to our system. Each region has a DSLAM (Digital Subscriber Line Access Multiplexer) device, and our proposed model is able to monitor and log the incoming traffic with the information of the region it is coming from. To satisfy this, our proposed system will be positioned in front of the internal network devices to catch the incoming requests from DSLAM devices. Our system will collect this region-based information and learn from it with unsupervised learning algorithms and mark potential infected regions in real-time. With the help of machine learning algorithms which are defined in Scikit-learn open-source library, we believe we can provide broad insights about the intrusion suspects.

To generate a proof of concept of the actual brain of our design, which is the Machine Learning Engine, we wanted to do some experiments on a real botnet data and we wanted to see if we can detect real botnets with the help of open source sci-kit learn library. To be able to detect intrusions, it is better to work on a dataset which contains real botnet behaviours instead of script-generated synthetic data. Therefore, in our study, we decided to work with a well-known dataset which is generated in Czech Technical University laboratories, CTU-13. The dataset provides multiple different scenarios, each simulating various malware infections. We had taken this dataset as a baseline, and we have cleaned and customized the data according to our scenario, and have applied ML algorithms on it to see how well we can detect the intruders.

To test the ML Engine which will be located as the brain of the system, we introduced some additional features, which will be calculated before the application of ML

algorithms such as; average inter-arrival time, average packet length, average data rate, same-length number of packets ratio, number of different protocol types. These features helped the ML Engine to obtain better accuracy.

MAKİNA ÖĞRENMESİ TABANLI AÇIK KAYNAK SALDIRI TESPİT SİSTEMİ

ÖZET

Güncel global ağ talep tahmin grafiklerini incelediğimizde, üstel bir artış trendi olduğu gerçeği yadsınamaz [3]. Bu yüksek hacimdeki veri talepleri, ağ operatörleri için kendi sistemlerini değişken taleplere cevap verecek şekilde tasarlama zorunluluğunu ortaya çıkarmaktadır. Bunun bir sonucu olarak da, söz konusu karmaşık ağ sistemlerinin güvenliğini sağlamak ya da bir atak sırasında hızla karar vererek davranmak daha zor bir iş haline gelmektedir.

Günümüzde her ne kadar ağ operatörleri davetsiz misafirlere karşı daha hazırlıklı olsalar da, saldırganlar da boş durmamakta ve hedef sistemlerin açıklarını bulmak için yeni yöntemler geliştirmekte ve yeni yollar keşfetmektedirler. En çok bilinen atak türlerinden bir tanesi de Servis-Dışı-Bırakma (Denial of Service - DoS) ataklarıdır. DoS atakları, hedef sistemdeki açıklara köle haline getirilmiş, internet bağlantısı olan cihazlar aracılığıyla saldırılarak yapılır.

Servis-dışı-bırakma atakları, hedef sistem kaynaklarını, meşru kullanıcılara ulaşılamaz hale getirilmeyi amaçlar. Bir servis-dışı-bırakma atağı, Komut ve Kontrol sunucusu tarafından köle yapılmış cihazlar tarafından yapılır. Dağıtık Servis Dışı Bırakma Atakları da bir çeşit Servis Dışı Bırakma atağı olup çeşitli lokasyonlardan çoklu köle cihazlarla yüklü miktarda trafik yaratarak hedef sistemin kaynaklarını tüketme yoluyla yapılır. Bu atakların verdiği zararlar, özellikle servis sürekliliği sağlaması gereken firmalar için geri dönülemez olabilir. Eğer firmalar, DoS ataklarına karşı korunma stratejisi belirlememişlerse, zararın ekonomik boyutu gerçekten büyük olabilir. Zira, DoS atakları, veri hırsızlığına, dolayısıyla da kullanıcı güveninin sarsılmasına sebebiyet verebilir.

Olası bir atağın ōncden belirlenebilmesi ya da tahmin edilebilmesinin yararları tartışılmaz. Ancak, ne yazık ki, özellikle de atak sırasında gerçek atak kaynağını tespit etmek neredeyse imkansızdır. Bunun sebebi, atak yapan bilgisayarların Internet Protokol (IP) adreslerinin gizlenmesidir.

Açıkça görülebilir ki, ateş duvarı (firewall) cihazlarının kurulması ve ağ kapasitesinin arttırılması gibi çözümler ataklardan koruyucu aksiyonlar olarak önerilmektedir. Ancak gelen trafik trendlerinin iyi anlaşılması, olası ataklardan korunmak için iyi bir yöntem olabilir [4]. Bu, gelen ve giden trafiğin sürekli izlenmesi ve hangi trafik tiplerinin normal ya da anormal olarak sınıflandırılabileceğinin belirlenmesiyle sağlanabilir. Bunun yanı sıra, [4], eğer genel trafik davranışları için bir baz çıkarılabilirse, istenmeyen trafiklerin en başında engellenebileceğini iddia etmektedir. Her ne kadar bu bahsedilen yöntemler, güçlü koruyucular olarak değerlendirilebilir olsa da, sistemler daima DDoS atak kurbanları olabilirler. Bu durum ise, akıllıca kurgulanmış, koruyucu ve hedef sisteme özel erken saptama sistemleri sayesinde aşılabilir.

Bu çalışmada, biz akıllı Makina Öğrenmesi tabanlı bir potansiyel atak saptama sistemi öneriyoruz. Bu sistem sayesinde, ağ operatörlerinin atak kaynağının lokasyonu hakkında bazı ipuçlarına önceden varabileceklerine inanıyoruz. Sistemimizi canlı bir trafik monitörleme sistemi olarak tasarlamaktayız, öyle ki bu sistem, gelen ve giden trafiği dinleyerek monitörleyecek ve bu veriden öğrenerek problemli istemcileri anormal olarak işaretleyecek. Bu şekilde, ağ operatörleri, mevcut ağ sisteminde olan biten hakkında geniş bir bilgiye sahip olabilirler.

Daha önceden bahsedildiği üzere, dağıtık bir atak sırasında, ağır yük altında olan bir ağda gerçek atak kaynağını bulmak, kaynakların kendisini gerçek kullanıcılar olarak gizleyebildiği gerekçesiyle neredeyse imkansızdır. Bu sebeple, atağın geldiği lokasyonla bağlantının kesilmesi kolay değildir. Ancak, eğer atağın geldiği lokasyon bilinebilir ya da tahmin edilebilirse, operatörler bağlantının kesilmesini sağlayabilir ve yükü azaltabilirler. Bu süre zarfında da atağın sistemlerine geri dönülemez zararlar vermemesi için çözümler arayabilirler.

Saldırı tespit sistemimiz, büyük miktarda veriyi işleyebilmek adına açık kaynaklı, ölçeklenebilir ve dağıtık olarak tasarlanmıştır. Sistemin özelleştirilebilir ve ölçeklendirilebilir tasarlanmasındaki amaç, önerilen sistemin yoğun veri trafiği olan ya da seyrek trafik görülen her türlü ağ sisteminin önüne kolayca kurulabilir olmasını istememizdir. Sistem, belirli zaman aralıklarında, Apache Kafka konularına işlenmemiş ağ verilerini toplayacak, Kafka tüketicileri ise bu işlenmemiş verileri işleyecek ve işlenen verileri Makine Öğrenmesi Motoruna gönderecektir. Makine Öğrenmesi Motoru, gelen verilere denetimsiz öğrenme algoritmaları uygulayacak ve sonuçları tekrar Kafka'daki ilgili konulara yazacaktır. Ayrıca bir Kafka tüketicisi olarak tanıttığımız Saldırı Sınıflandırıcısı, şüpheli verilerin gerçekten kötü olup olmadığına karar verecek. Saldırı Sınıflandırıcısı'nın kararından sonra, sonuçlar sistem yöneticilerinin kullanımı için Elastic Arama'ya yazılacaktır. Sonuçları sunmak ve görselleştirmek için Kibana arayüzleri kullanılacaktır.

Bizim senaryomuzda, N tane sistemimizle konuşmak için izinli bölge olduğunu varsayıyoruz. Her bir bölgenin kendine ait Digital Abone Hattı Erişim Çoklayıcı (DSLAM - Digital Subscriber Line Access Multiplexer) cihazı vardır. Önerdiğimiz model, gelen trafiği bölgesi ile birlikte günlüğe kaydedecektir. Sistemimiz, günlüğe kaydedilen gelen trafik verisinden, makina öğrenmesi algoritmaları sayesinde modeli öğrenecek ve potansiyel köle cihaz barındıran bölgeleri işaretleyecektir. Makina öğrenmesi teknikleri sayesinde, şüpheli istemciler konusunda geniş bir kavrayış sağlayabileceğimize inanmaktayız.

Örnek bir konsept kanıtı için gerçek köle cihaz davranışları içeren bir veri seti ile çalışmak daha gerçekçi sonuçların alınmasını sağlayacaktır. Köle bilgisayar saptama için kullanılan data setleri genelde sentetik olarak bilgisayar ortamında yaratılmaktadır. Bilgisayar ortamında yaratılan veri setini kullanmayı tercih etmedik, zira bu şekilde sentetik data ile yapılan çalışmaların realist sonuçlar vereceğine inanmamaktayız. Çalışmamızda Çek Teknik Üniversitesi laboratuvarlarında hazırlanmış, çoklu kötücül yazılım ve atak çeşidi içeren CTU-13 veri setini kullandık. Bu veri setini kendi ihtiyaçlarımıza göre özelleştirerek üzerinde makina öğrenmesi algoritmaları çalıştırdık.

Sistemin beyni olarak konumlandırılacak olan Makina Öğrenmesi Motorunu test etmek için, makina öğrenmesi algoritmalarının uygulanmasından önce hesaplanacak

bazı ek özellikleri sunduk; ortalama varış arası zaman, ortalama paket uzunluğu, ortalama veri hızı, aynı uzunluktaki paket sayısı oranı, farklı protokol tipleri sayısı. Bu özellikler, Makina Öğrenmesi Motorunun daha iyi doğruluk elde etmesine yardımcı oldu.

1. INTRODUCTION

1.1 Purpose of Thesis

According to Cisco's 2018 Annual Cybersecurity Report [5], 53% of the security attacks resulted in cost of $500.000 or more to the organizations. Considering the fact that the attackers are developing their techniques and skills for intrusions, we might expect a lot more damage to come. Therefore, organizations and companies, especially the ones which have large number of active users, such as banks, e-commerce sites etc. need to have intelligent intrusion detection systems, preferably customized for their network infrastructure.

In the light of the aforementioned reasons, in this work, we focus on one of the most difficult attacks to avoid and defend, for the security experts; Distributed Denial of Service (DDoS) attacks. Distributed attacks make companies inoperable for long periods of time; and it is crucial to shutdown the connections of attackers as fast as possible to minimize the service denial.

It is hard to trace back to the origin of the attack in DDoS concept, however, we believe that, logged network traffic data can give many clues to find the possible attack source locations. If we analyze the network traffic data and observe the normal traffic behavior with applying Machine Learning (ML) techniques; we can detect abnormal traffic and we can have an idea of from where the possible DDoS attacks might be coming.

1.2 Background

1.2.1 DDoS attacks

A DDoS attack is a coordinated attack using a huge number of compromised hosts [6]. A DDoS attack consists of an attacker, multiple controllers, multiple slave computers and a victim as seen in Fig.1.1.

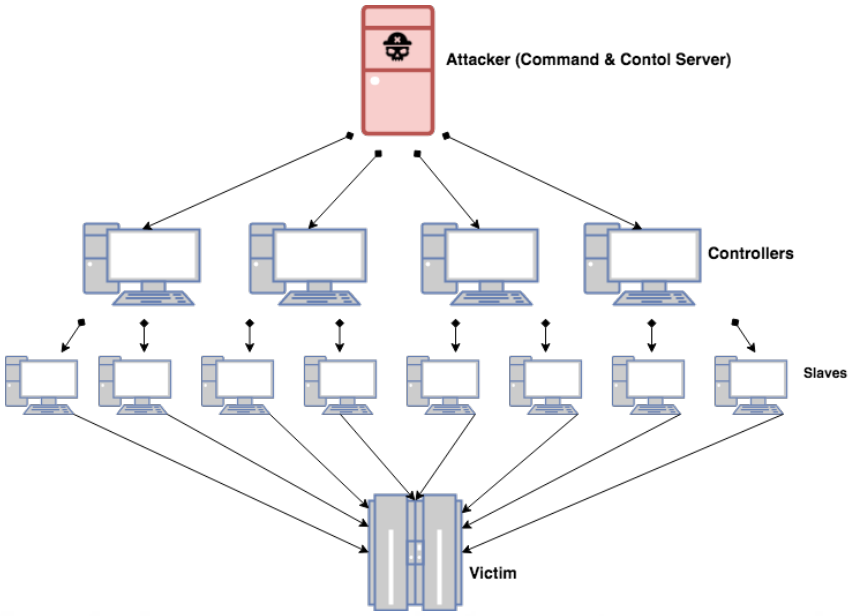


Figure 1.1 : Layered architecture of a DDoS attack.

Attacker: Attacker is the source, who wants to exhaust the resources of the target machine with simultaneous requests in a very short time. Attacker does this either by forming or hiring a bot network via installing a bot software on it. A DDoS attack has four elements; attacker, one or more controllers or handlers, slave computers and one or more victims.

Controller: Attacker mainly controls the controller machines i.e. handlers to set up the Bot software (DDoS attack software) to slave computers.

Slave Computers: These are the compromised computers, which has Bot software to generate stream of packets to deny the services of the victim computer.

Victim(s): The target computer which is under a DDoS attack.

It is very hard to trace the origins of a DDoS attack because of the following reasons:

1. Compromised hosts send packets with their spoofed source IP.
2. The layered architecture of the DDoS attacks, makes it difficult to find the real source; because the true source is not the slave (daemon) computers or handlers.
3. Since the host computers are replicated and distributed, even one of the slave computers found and the communication is shutdown, the other slaves can continue the attack [7].

1.2.2 Intrusion detection systems (IDS)

An Intrusion Detection System (IDS) is a software or a physical device which detects and reports malicious activities by monitoring system or network activities in an information system. There are several studies conducted and many frameworks introduced as IDS. A study in this area is conducted by DARPA initiative at 1998. Staniford-Chen et. al. introduced 'Common Intrusion Detection Framework' (CIDF) [8] as an exemplary architectural overview of a common IDS framework. A CIDF is a four-element framework as illustrated in Fig. 1.2.

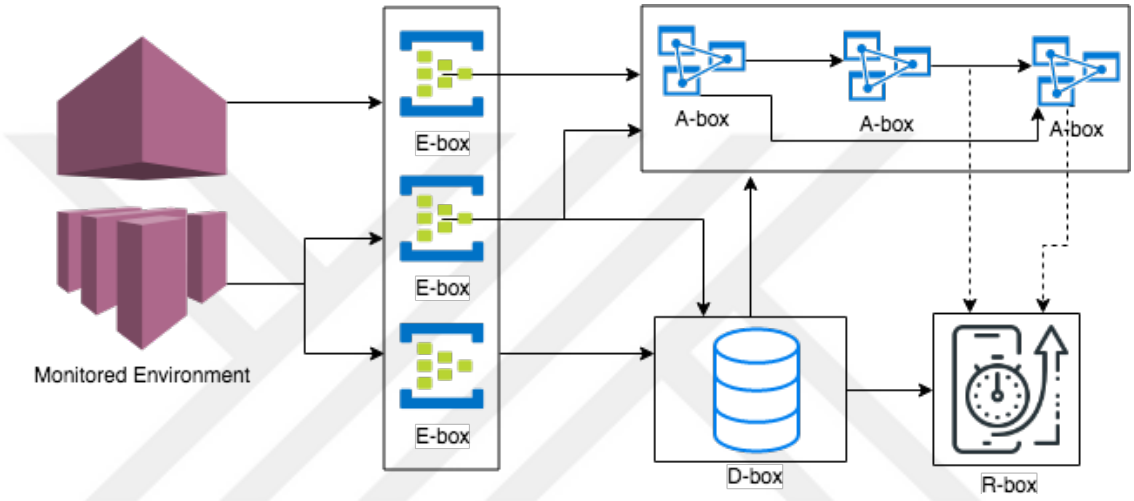


Figure 1.2 : Illustration of a Common Intrusion Detection Framework.

CIDF communicates through modules via message passing and it has four modules as explained below:

- Event generators ("E-boxes"): These generators acquire information for further analysis of other modules of CIDF.
- Event analyzers ("A-boxes"): These modules analyze events and detect potential malicious activities and feed D-boxes and R-boxes accordingly.
- Event databases ("D-boxes"): These modules stores elements for persistency where necessary.
- Response units ("R-boxes"): When an intrusion is detected, these units carry out some actions such as disconnecting servers, killing processes or changing file permissions etc.

Our proposed IDS has the three of aforementioned modules, which are E box, A box and D box; however, we do not have an R-box in the traditional sense, since our system does not provide an alarm module. At the end of our detection process, we write the results to our D-box, however we do not have a proactive alarm system which directly alerts administrators and acts accordingly by killing processes, ceasing connections etc.

An IDS can be classified in two classes according to execution strategy; a general IDS can be either network-based or host-based. A host-based IDS (colloquially HIDS) is integrated into a host device, and monitors operating system level misuse, such as file activities etc. A network-based IDS (NIDS) monitors network activities of an information or communication system. An NIDS analyzes IP information, packet headers, network data volume etc. to detect malicious activities. Our proposed model can be classified as a network-based intrusion detection system.

In addition to execution strategy, we should also mention about the cyberanalytics techniques which are used for an IDS varies in three types; misuse (signature) based, anomaly based and hybrid techniques which constitutes from a combination of misuse and anomaly based techniques [9]. With signature-based techniques, operators can detect known type of attacks with the help of the information of unique signature byte sequences or some pre-defined rules. This technique is advantageous because it is hard to produce false positive results. However, signature-based techniques have some drawbacks such as they cannot detect zero-day attacks. On the other hand, anomaly-detection techniques are advantageous in that sense, because they focus on the effects of an attack rather than its characteristics and known behaviours. But one main disadvantage of anomaly detection techniques is that it is possible for them to categorize previously unseen legitimate activities as malicious. Therefore, it can be said that anomaly detection techniques can have high false alarm rates. A disadvantage of both techniques is that if an attacker understands the behavior of the installed IDS, then they can change their attack strategies and signatures, and they can be a zero-day attack candidate. Our proposed system model uses anomaly detection techniques.

1.3 Literature Review

Cybersecurity is a famous paradigm, which uses technology to protect individuals, organizations and networks from cyber attacks [10]. There are many ways to protect systems from digital attacks, such as deploying firewalls, adding restrictions to the certain ports, employing Access Control Lists (ACL), using the advantages of antivirus tools or utilizing from Intrusion Detection Systems (IDS). Especially, IDS's are extensively used, because they provide an early alarm system and can protect the networks from many attack types such as zero-day-exploits, denial-of-service attacks etc. When it comes to certain kind of attacks such as DDoS attacks, there are numerous recommended actions to take as precautions when a network is a victim of the attack. Distributed Denial of Service attacks, use vulnerabilities of a system by sending loads of traffic to exhaust the system resources and to make the system unavailable. As it is known that the consequences of a DoS attack might be very serious, the researches on the field of early anomaly detection systems picked up the pace recently. Especially, detecting abnormal behaviors with the help of machine learning approaches is very appealing topic, as ML based IDS's can learn from the previous traffic data behaviours and mark the outliers intelligently.

Anomaly detection or outlier detection -as in machine learning paradigm-, is a method which eliminates the anomalies from normal behaviors. Anomaly detection as part of DDoS attack protection helps network operators to catch abnormal botnet-like behaviors which deviate from usual normal behavior. Anomaly detection based IDS's are advantageous because they can be customized according to the network so that the intruders are not able to find out which of their activities can be left undetected [9].

In [11], the authors present a detailed work about anomaly detection techniques and challenges. The work also gives broad insights about pattern categorization with machine learning. Since ML techniques are focused on generating a model whose performance is increased according to the previously learned behaviours, they are highly desirable because of the ever-changing network traffic behaviours. However, authors also state that there might be some drawbacks of machine learning approaches, which some of them might require high amount of computational work.

Lippmann and Cunningham [12], proposes an intrusion detection system with specific keyword selection approach by the help of neural networks. They reduced the false alarm rates by achieving 80% of true intrusion detection.

In [13], the authors presents NETMINE framework, which identifies the patterns in the network traffic data with the help of data mining techniques. NETMINE extracts rules for anomaly detection with generalized association rules such as subnet traffic. Their work showed that NETMINE framework is very successful at classifying the network patterns.

In addition to the aforementioned studies, which focuses on the techniques rather than developing an actual platform, there are also multiple Network Based Intrusion Detection Systems available both commercially or in the phase of development which will be served as open source products by many university laboratories.

One IDS for anomaly detection is Siren [14], which is designed by Penta Security. Siren injects crafted virtual human input to the actual benign user inputs in order to detect mimicry attacks. They succeeded to find mimicry attacks of ten different spyware programs.

Another behavior based anomaly detection system is provided by Symantec, which is an Intrusion Protection System. The system is customizable by letting clients create their own signatures.

AirDefense IDS is another product which uses context-aware detection by providing monitoring in real time.

There are also studies conducted by some universities and laboratories. For example, Autonomous Agents for Intrusion Detection by CERIAS/Purdue University is an open platform, which satisfies distributed anomaly detection. It is one of the good examples for decentralized Intrusion Detection Systems, which are very rarely seen. Unfortunately, the study has been terminated indefinitely.

Another good open platform is NFIDS (Neuro Fizzy Intrusion Detection System), uses neural network and fuzzy logic for intrusion detection.

All of the studies perform very well with their specific cases. However, none of them solves the limitations of centralized detection systems, e.g. scalability, alongside with machine learning-based approaches.

In our study, we provide, vendor-specific, completely open-source, scalable, machine-learning based, anomaly detection system which can be customizable by the needs of client system. And also, our system can work seamlessly, even when one of its parts become disfunctional, because the whole system is designed to be seamlessly degradable.

2. SYSTEM ARCHITECTURE

In this work, in an effort to detect or predict the locations of slave computers, we propose a novel, distributed, scalable, machine-learning based Intrusion Detection System (IDS). Our system can handle and process huge volume of data with low-latency. The Machine Learning Engine (ML Engine) in our IDS, continuously learns from real-time data and make predictions accordingly.

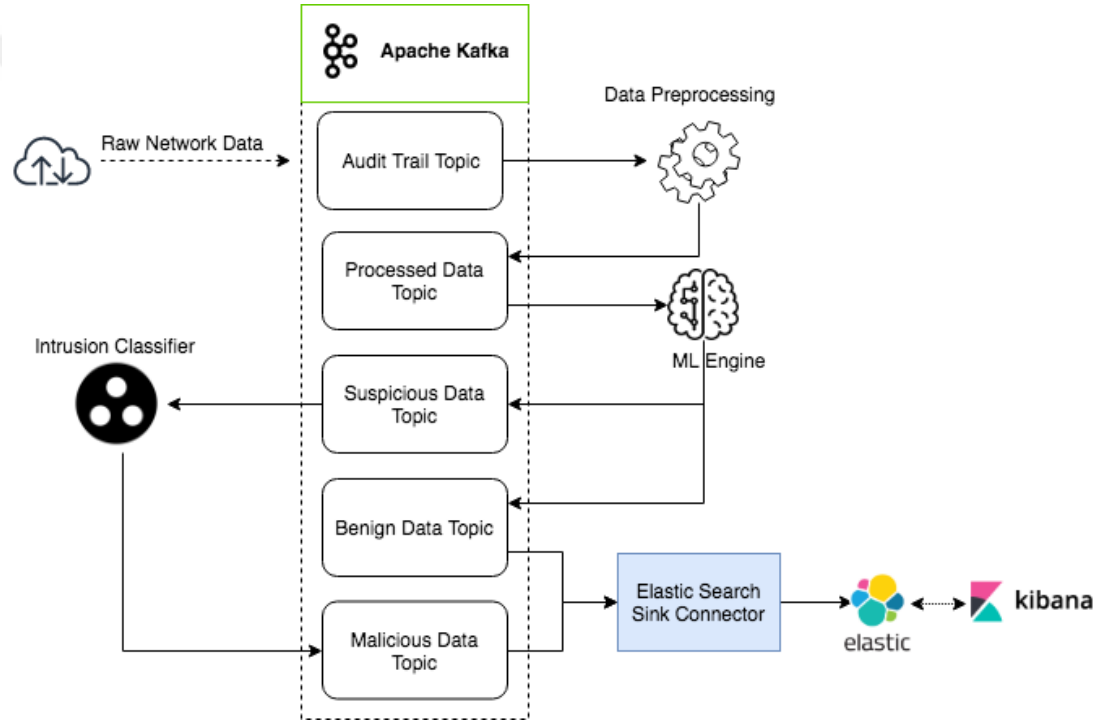


Figure 2.1 : System model of proposed IDS.

Figure 2.1 shows the proposed system architecture for our intrusion detection platform. We use Apache Kafka to scale and process the incoming data in parallel, and we introduce multiple topics under it.

Data Preprocessing, ML Engine and Intrusion Classifier are the modules, which directly interact with the Kafka topics. At the edge of our platform, we put an Elastic Search based logging system. The machine learning results will be an input to Elastic Search engine and will be released via Kibana interfaces for the use of the system

and network administrators to query anomalies. The following sections give detailed information about the components of our proposed system.

2.1 The Raw Network Data

Our proposed model gathers incoming network requests and logs them to Audit Trail Topic in Apache Kafka without doing any further data manipulation. The raw network data is going to be processed in accordance with OSEMN (Obtain, Scrub, Explore, Model, Interpret) pipeline, with the help of Data Preprocessing and ML Engine Modules.

Each request entry in the Audit Trail Topic should have some mandatory information such as request timestamp, protocol, flow duration and information related to source host, etc. in order to apply our region-based anomaly detection model.

As a proof of concept, we are going to use an experimental dataset for detecting anomalies with machine learning algorithms. With the demonstration of the ML algorithms on the exemplary dataset, we also aim to show that our proposed model has potential as a promising platform to be applied on real-world scenarios. More detailed information about the dataset will be provided in Section 3.1.

2.2 Audit Trail Topic

In our proposed system, the incoming data will be written to Audit Trail Kafka Topic without being further processed. Data Preprocessing Module will "Scrub" the data. Scrub is the second step of the OSEMN framework, which the cleaning and the necessary calculations are carried out. The incoming network traffic will be examined in 10-minute time windows. The reason being is that the examination of 10-minute windows will be sufficient for detection of anomaly characteristics.

To keep 10-minute-window logs, we introduced a set of 6 partitions in Audit Trail Topic, such as Partition_0-10, Partition_10-20, Partition_20-30,...,Partition_50-60. The network flow logs will be written to these 6 variants of Kafka partitions according to their timestamps. For instance, Partition_0-10 holds the flow information which arrived to the system in the first 10 minutes of each hour. In the same way, Partition_50-60 will hold the last 10 minute of each hour's audit logs. When a flow

arrives to the system, the corresponding topic will be decided via help of a basic modulus operation over 10 on the flow timestamp. For example, a flow which arrives on 68th minute of the daily system start, will be written to Partition_0-10.

After properly logging into the partitions of Audit Trail Topic, the message reading and data processing parts should be designed. To this end, Apache introduced Consumers which are used for reading messages from Kafka Topics [15]. In order to read messages from Kafka Topics and then later work on them, it is required to define a consumer object and make this object subscribe to the topic. However, in our case, we want the consumption from our topic to be scalable and made in parallel. For this reason, we introduced three consumer groups, which consumes different partitions as seen in Figure 2.2. As it can be seen from the figure, consumers are scattered across partitions scattered. The reason behind this is that we want our data processing made in parallel and we want to keep the consumer idle time at a minimum level. For instance, let us think a scenario where we assign Consumer_1 object to Partition_0-10 and Partition_10-20. In this case, the processing of the messages in Partition_10-20 will start until Consumer_1 reads and validates all messages in Partition_0-10. In that case, there would be no meaning to define multiple consumers as there would be no way to process reading messages in parallel. Despite defining one consumer can be

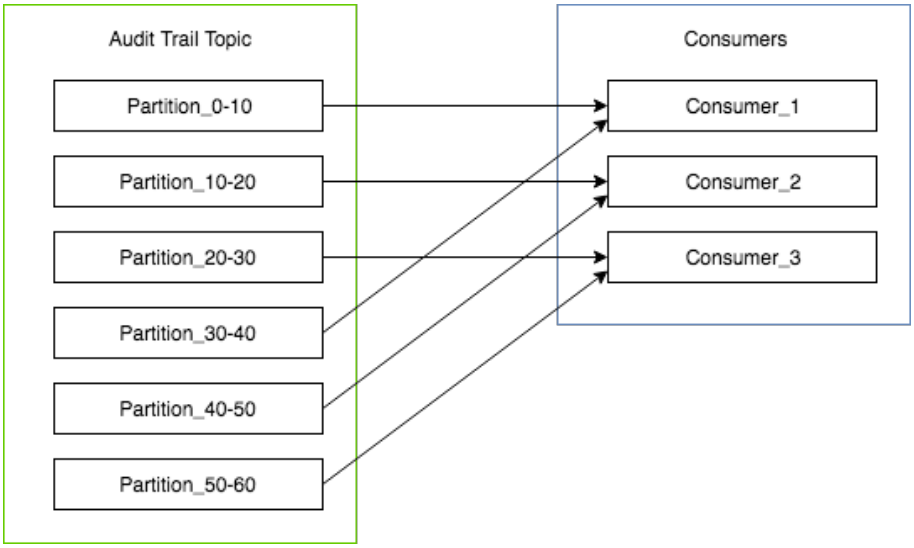


Figure 2.2 : Relationship between audit trail topic and consumers.

advantageous for some cases, in our case, since we expect an extensive amount of incoming requests, we can fall into a case that producers create messages more than our consumers can meet and process. For this reason, we are going to use more than

one consumer for each partition. These consumer groups can be *rebalanced* in case of a need, in other words, it is possible to change ownership of partitions or add other consumers to avoid single point of failure.

2.3 Data Preprocessing Module

This module is the engine who consumes and validates Audit Trails and performs a *Scrub* operation on them. After cleaning and creating the required features for the network data, Data Preprocessing writes the results to Processed Data Topic to be an input for ML Engine. In our proposed model, we assume that every individual audit

Table 2.1 : List of fields in Audit trails of proposed framework.

Field Name	Data Type
Timestamp	DateTime
Flow duration (in seconds)	long
Protocol Type	string
Source IP Address	string
Source Port	int
Destination IP Address	string
Destination Port	int
Total Packets	int
Total Bytes	long
Region	string

trail has all the information defined in Table 2.1, together with the Region information which shows that where the flow is actually generated from. Some of these fields in audit trails will be used in the calculation of additional model features, some of them will also be an input to our models as features and the remaining fields are redundant and will be dropped. Each distinct flow should have an identifier, so that machine learning models can discriminate them as legitimate or malicious. Using only source IP addresses as the identifier, is not enough for understanding the real outliers. For that reason, we will keep a HashSet of flow identifiers (i.e. Flow Identifier HashSet) alongside with features which the machine learning algorithms will be applied on. A flow identifier is composed of the following fields similar to introduced in [16] excluding the Protocol Type field and with an additional Region field as shown in Figure 2.3. However, we are not going to hold these identifiers as plain text tuples. Contrarily, we will calculate MD5 hashes for these identifiers and keep them in our Processed Data Topic with those hashes to apply machine learning easily. Alongside

Flow Identifier				
Region	Source IP	Destination IP	Source Port	Destination Port

Figure 2.3 : Fields of flow identifier.

with hashes of identifiers, The Flow Identifier Hashset data structure, will also keep a timestamp which identifies the current time-window. These timestamps will be used to age the flow identifier hashes after a certain period of time. This aging time period shall be decided by the system and network administrators so that the aged hashes can be moved from the Suspicious and Benign Data Topics, in order to give more accurate decisions which are reflecting the current system situation more realistically. In addition to the fields in audit trails, our Data Preprocessing Module will also calculate the following features which the details are given in Section 3.3.2: Average inter-arrival time, average packet length, average data rate, same-length number of packets ratio, distinct number of different protocol types.

2.4 Processed Data Topic

Cleaned and processed data will be written to Processed Data Topic, which is going to be consumed by the Machine Learning Engine of our platform. An exemplary record for Processed Data Topic is shown in Figure 2.4 .

Flow Identifier: 889d6c07d1985f0a277100e19c5a4ba6	**Protocol Type:** tcp	
	Distinct Number of Protocol Types: 1	
Average inter-arrival time: 10s	**Region:** R1	**Average Packet Length:** 3480bits
Average data rate: 300bits/sec	**Same-length Number of Packets Ratio:** 0.3	

Figure 2.4 : ML ready data fields after preprocessing.

2.5 ML Engine

Machine Learning Engine is the brain of our proposed framework. It consumes the messages which are kept in processed data topic, and applies unsupervised ML algorithms on them. The ML results are written in two different topics according to their characteristics; Benign Data Topic and Suspicious Data Topic. These two topics should be synchronized with each other, in other words, if a flow is marked as benign and written in Benign Data Topic, then its corresponding flow identifier should not be in Suspicious Data Topic. If our algorithm marks a flow as outlier, this is not directly treated like it is a malicious flow. These flows are written into Suspicious Data Topic for further evaluation.

2.5.1 Feature selection

Feature selection is one of the most significant process, which specifies the success of ML-based anomaly detection. To have better anomaly detection accuracy, the set of features which ML algorithms are going to be applied upon, should be chosen wisely. Nevertheless, choosing the best set of features never means that using all of the features that could be extracted is beneficial. If extensive number of features are used, overfitting and decreased accuracy are a few of many problems that researchers might face.

- **Average inter-arrival time (λ)** [17]: The interval of benign connections is usually longer than the connections opened by botnets [18]. Therefore, the frequency of the incoming packets in a time window, is an important metric for us to discriminate the legitimate and infected hosts. The inter-arrival time is calculated as $\lambda = T_w/N_p$, where T_w is the size of time window in seconds, and N_p is the number of incoming packets which belongs to one specific host IP.

- **Average packet length (L)** [17, 18]: This value shows us the size of packets the system receives from a specific host in a time window. This value is typically larger for legitimate traffic, since the number of packets for legitimate hosts are at an ordinary level. Average payload packet length is described as the total packet size in bits over the number of packets in a time window. $L = S_p/N_p$, where S_p is the

total size in bits arrived in a time window, and N_p is the number of incoming packets which belongs to one specific host IP respectively.

- **Average data rate (D)** [19–22]: Average data rate is the received number of bits per second in a time window. $D = S_p/sec$, where S_p is the total size in bits arrived in a time window. This metric is also used for distinguishing IRC traffic.

- **Same-length number of packets ratio** [17]: This is a ratio which defines the ratio of the total packets which are of the same length over the total packets in a time window. In legitimate windows, this parameter is expected to be small per each flow.

- **Number of different protocol types**: Distinct number of protocol types is an indicator that gives information about the characteristics of legitimate/malicious hosts. Typically, 1 or 2 different protocol types is used per host. [17]

2.5.2 Label encoding

In this module, **Protocol Type** and **Region** fields are label-encoded for further processing. Label encoding is a transformation technique which normalizes labels. In machine learning, there is no way to use string values to run models on. This technique can only be applied when there is a finite number of classes on the corresponding label. Otherwise, models will throw "y contains previously unseen labels" error on transformation process. In our IDS, we need to transform some string values to their numerical representations. For this purpose, we used *LabelEncoder* utility class of sklearn library under *preprocessing* namespace.

2.5.3 Preparing training and testing sets

In machine learning applications, it is very important choosing the right training, testing and validation data sets. Training data is high-coverage, partial data set of the whole data, which the model learns from it. After the learning phase, the test data is used to make predictions. Validation data is used for hyperparameter tuning and it is optional, however, in our model, we do not use a validation dataset.

Both training and testing data sets must have adequate amount of relevant samples of the properties which are planned to be studied and predicted. Therefore, the operation of splitting the whole data set into two data sets is quite critical.

Splitting the data set as a ratio of 8:2 training:testing is very common. There is a general opinion about having more training data means getting better models [23], and there are some studies and propositions against this opinion [24]. By taking into consideration both of the opinions, we think that it is more suitable for our model to use 8:2 ratio as shown in 2.5, for training:testing data sets.

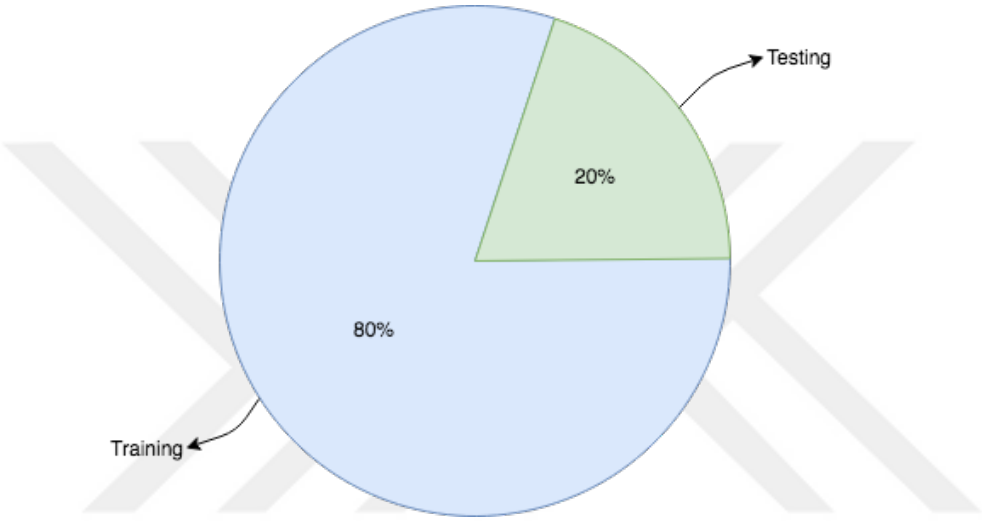


Figure 2.5 : Ratio of train:test data used in our system model.

In this phase, we utilize from Scikit-learn, which is also an open-source, powerful Python based machine learning library. We will use `train_test_split` method of `model_selection` class. An example call with required parameters is shown in Figure 2.6.

```
import pandas as pd
from sklearn.model_selection import train_test_split

data = pd.read_json("partition_0-10.log")
data_train, data_test = train_test_split(data, test_size=0.2,
train_size=0.8, random_state=None,shuffle=True,stratify=None)
```

Figure 2.6 : Split train and test data using Scikit-learn.

The below listing describes the parameters which needs some explanation, the other parameters used in `train_test_split` are self explanatory.

- **`random_state`**: This is the seed which is going to be used as a seed for the random number generator (RNG). It is an optional parameter, when it is None, RNG

uses the default seed of `np.random` which will try to read from /dev/urandom if it is defined, or it will use the seed from the clock. [25]

- **`shuffle`**: This parameter is used to shuffle the given dataset before splitting it into training and testing. We used True to achieve cross validation.
- **`stratify`**: This parameter ensures that the data split is done in a stratified way. None is the default value.

2.5.4 Apply machine learning algorithms

After the end of the preparation phase of testing and training sets, we are going to apply machine learning algorithms which are part of Scikit-learn library. Since we are going to have a real-time data, and we assume that we do not have the knowledge of previously-seen abnormal values, we will apply unsupervised learning algorithms for outlier detection. We will not use novelty detection algorithms, because at any timestamp, we cannot make sure of we have a clean dataset which is not contaminated with abnormal values.

Our Machine Learning engine will apply three different outlier detection algorithms and it will send suspicious data to Suspicious Data Topic and normal data to Benign Data Topic to be further processed by Intrusion Classifier.

2.5.4.1 Isolation forest

Isolation Forest algorithm uses Random Forests to classify the data and detect outliers. First, it selects a random feature from feature set and then selects a random splitting value which is in between minimum and maximum seen values of the candidate feature, and then isolates those observations around this selected number as illustrated in 2.7. One of the advantages of using random forests is that they use multiple trees to decide, this behavior reduces the risk of overfitting.

Another advantage is that training times in random forests are less than the other algorithms. Also they can work really good with large sets of data. The basic idea behind the random forest algorithm is that during the training phase, it constructs multiple decision trees and at the end, the majority of the decision trees is chosen

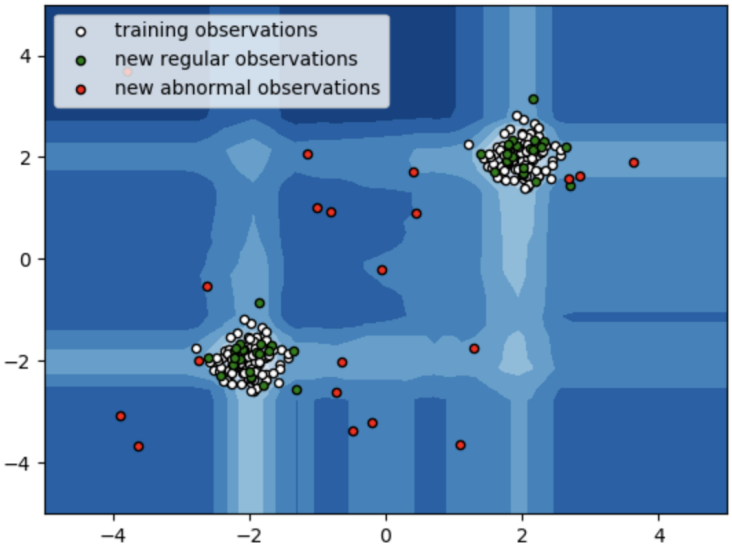


Figure 2.7 : Graphical representation of isolation forest [1] [2]. Retrieved from *scikit-learn 0.21.3 documentation.*

```
from sklearn.ensemble import IsolationForest
from pandas import Series

isof_model = IsolationForest()
isof_model.fit(data_train)
Series(isof_model.predict(data_test)).value_counts()
# value_counts() will give two value sets,
# which are total count of inliers and outliers;
# as inliers labeled with 1, and outliers labeled with -1*/
```

Figure 2.8 : Application of Isolation Forest algorithm in scikit-learn library.

as the final decision. At the end of Isolation Forest phase, the decisions will directly be sent to Intrusion Classifier for further processing.

2.5.4.2 Elliptic envelope

Elliptic Envelope (EE) routine assumes that the training data comes from a Gaussian distribution. The algorithm, models the data with covariances between the features. It tries to estimate an ellipsis which contains majority of the data by using FAST-Minimum Covariance Determinate (FMCD) [26]. The data outside of this ellipsis is decided as outliers as illustrated in 2.9.

One hyper-parameter that EE requires is **contamination** value, which cannot be known with high-accuracy before the algorithm is run at least once [27]. This hyper-parameter basically defines the expected the rate of outliers, in other words, it defines how much of the data can be outside of the highest dimensional ellipse which contains the majority of the input data.

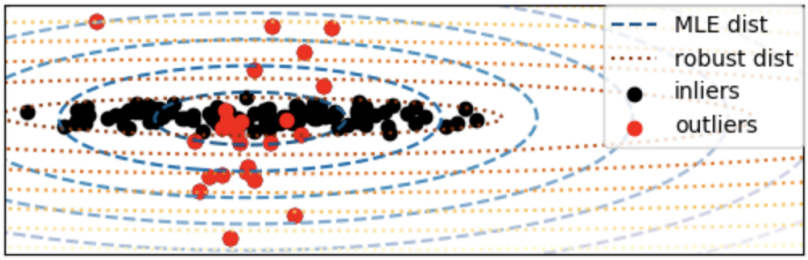


Figure 2.9 : Mahalanobis distances of a dataset after the application of Elliptic Envelope routine [1] [2]. Retrieved from *scikit-learn 0.21.3 documentation*.

To be more specific, FMCD gets some small samples from the input data and computes the mean value $\overrightarrow{\mu}$ and builds a covariance matrix C, for each feature dimension. After that Mahalanobis distance (d_{mh}) is computed for every single data row (data vector $\overrightarrow{x}$) in each sample as given in the following formula:

$$d_{mh} = \sqrt{(\overrightarrow{x} - \overrightarrow{\mu})^{\mathrm{T}} C^{-1} (\overrightarrow{x} - \overrightarrow{\mu})} \tag{2.1}$$

d_{mh} of the every vector in subsamples are iteratively computed, until $detC$ converges. An ellipse is created from the C which has the smallest determinate from all of the samples, and as aforementioned before the outliers are determined according to this ellipse.

```
import pandas as pd
from sklearn.model_selection import train_test_split

data = pd.read_json("partition_0-10.log")
data_train, data_test = train_test_split(data, test_size=0.2,
train_size=0.8, random_state=None,shuffle=True,stratify=None)
```

Figure 2.10 : Application of Elliptic Envelope algorithm in scikit-learn library.

2.5.4.3 Local outlier factor

When the subject comes to outlier detection methods, classification algorithms come to the rescue. Some classification algorithms operate over the idea of having one centroid. But however, most of the time it is not enough to find all of the outliers, For example in Fig 2.11, a centroid based approach will detect A and B as outliers, and C as inlier. However, in reality, B and C are outliers but A is not.

The methods like Elliptic Envelope can be more precise to find extreme outliers, however, it also might not satisfy full coverage for finding abnormalities. For all

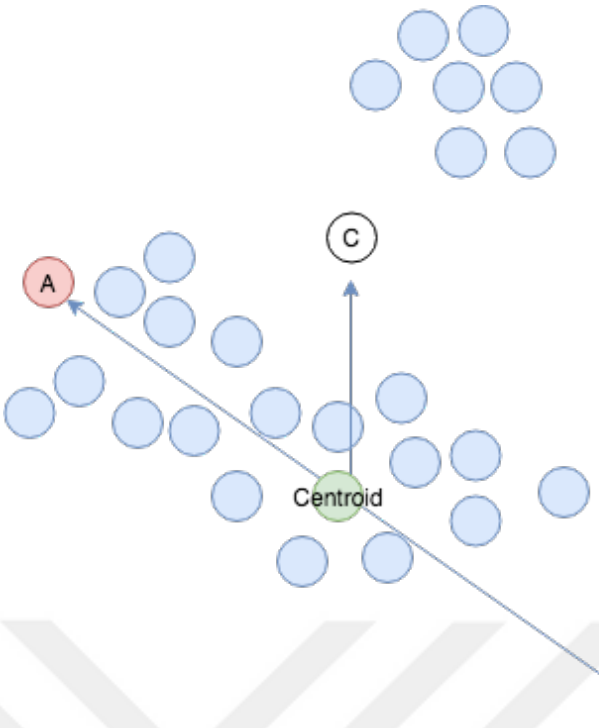


Figure 2.11 : Distance based outlier detection approach.

of the aforementioned drawbacks could be overcame by the help of Local Outlier Factor (LOF) algorithm. This algorithm is a density-based routine, which computes a Local Outlier Factor and it indicates a score of the anomaly. The basic idea behind this algorithm is that the abnormal observations have lower density than the normal observations.

LOF score of a point is deducted from k-nearest neighbors (kNN). For an observation, it is equal to the average local density of kNN over local density It is expected that an outlier data point has much smaller local density, compared to the normal points. If the LOF score of an observation is less than 1, this point is not an outlier otherwise it indicates that the observation is an outlier.

```
from sklearn.neighbors import LocalOutlierFactor

lof_model = LocalOutlierFactor(n_neighbors=35, contamination=.15)
lof_model.fit_predict(x)
scores = lof_model.negative_outlier_factor_
print(-scores)
```

Figure 2.12 : Application of Local Outlier Factor algorithm in scikit-learn library.

2.6 Intrusion Classifier

Intrusion Classifier will consume the messages in Suspicious Data Topic. The Intrusion Classifier will give the final decision for a region to host an infected client or not. The decision will be given by comparing the accuracy results or scores of the applied Machine Learning algorithms. We introduce a novel Regional Infection Coefficient (RIC) parameter, which can be customized by the system and network administrators. This coefficient is a pre-defined sensitivity number, which depends on each region's usual traffic characteristics. High numbers of RIC will catch more extreme infections, thus making the region less sensitive to changes. If a region always generates huge amount of traffic, then this threshold should be set to a low value, in order to mark malicious traffic easily. RIC can be set high for a region, for example, if the following characteristics are observed for that region:

- The hosts from this region always generate small-sized packets (e.g. between 50-300 bytes)
- There are typically 1 or 2 distinct protocol types seen from this region

2.7 Malicious Data Topic

After IC finishes its job, if it decides any host as infected, the flows belonging to that host alongside with its region informations are written to Malicious Data Topic. The data in this topic, alongside with Benign Data Topic will be written to indexes in Elasticsearch for further analysis for network and system administrators.

2.8 Elastic Search & Kibana

Elasticsearch is a widely-used, scalable search engine. The data is stored in Elasticsearch in JSON (Javascript Object Notation) documents. We will also introduce Kibana interfaces, which is a data visualization plugin for Elasticsearch. With Kibana, one can analyze, search and visualize the data. Using Elastic Stack (a.k.a ELK Stack), will be very beneficial for network and system administrators, allowing them to see the system health from a broader point of view.

In order to move data from Apache Kafka to Elasticsearch, we need Elasticsearch connectors. There are two types of connectors: sink connectors and source connectors. Source connectors collect data from external systems into Kafka topics, while sink connectors delivers data from Kafka to external systems [28]. So for our purpose, we need a sink connector. Kafka Sink Connector will take data from our Kafka topics which are Malicious Data and Benign Data topics, and writes this data to an index [29] in ElasticSearch. An important property of the connector framework is that it also provides a REST (Representational State Transfer) API for managing configuration and makes it really easy to use and set up. One available open source connector that we can use for this purpose is Kafka Connect ElasticSearch product, which was developed by Apache Kafka creators. It uses HTTP (Hyper-Text Transfer Protocol) based Elasticsearch client library [30], which makes the API compatible with older/newer versions of Elasticsearch.

3. PROOF OF CONCEPT STUDIES

In order to test the applicability of our proposed ML Engine, we made a couple of Proof of Concept (POC) experiments. In this part of our work, we tried to demonstrate that an open-source ML library can be a good candidate for a low-cost, scalable and applicable IDS system. Our experiments showed that ML algorithms in Scikit-learn library can be used for outlier detection, and it is possible to build a completely open-source, scalable, device/vendor-independent botnet-warning system for system and network administrators. For the sake of prototyping, we wanted to use a contaminated dataset which has botnet traces alongside with benign data. In the following section, we give detailed information about our choice of dataset.

3.1 The Anatomy of the Dataset

The key to develop a successful machine learning application is to work on a qualified, clean and representative data. Experiments should be done on subsequent amount of samples. If experiments are going to be done on synthetically generated data, then this dataset should be very similar to the real life case which is planned to be simulated. Also, the dataset should be unbiased and clean, such that the applied algorithms on this data should result in very small ratio of false positives (FP).

There are many datasets available for botnet detection experiments. Many universities and laboratories provide synthetically generated datasets, and some educational institutions also share some attack-data which are generated in their internal laboratories. It is important to use a dataset which contains real-world botnet data, rather than network traces which are generated synthetically [31]. The selected dataset should provide an adequate level of heterogeneity to successfully simulate a live network [18].

Because of the aforementioned reasons, we have chosen well-known CTU-13 dataset, which was generated by Czech Technical University (CTU). The dataset is fairly a new dataset, and it contains real botnet traffic flows, alongside with background and normal

traffic. CTU-13 has thirteen different captures as known as scenarios, which each of them consists of different malwares using different protocols.

For our experiments, we have used Scenario 10, because it has the most infected hosts i.e botnet flows, and it simulates UDP DDoS attacks. Table 3.1 gives the detailed information about Scenario 10 data.

Table 3.1 : Scenario 10 characteristics of CTU-13 dataset.

Botnet Name	Duration (h)	Infected Hosts	Protocol	Attack Type
Rbot	4.75	10	IRC	UDP DDoS

3.2 Our Approach for Botnet Detection

We propose a machine learning based approach to detect anomalies in a live network. Our ultimate aim is to provide an early intrusion detection system by predicting the possible botnet locations before a DDoS attack occurs. During an attack, it is quite troublesome to locate the actual locations of the botnets and cease the existing connection in between. In order to be able to take action promptly, it is important that we have sufficient audit trails and we have the knowledge of expected average system load at a specific time or from a specific region. We believe that if the system administrators have the knowledge of the type and quantity of the traffic which their system receives ordinarily, they can be alerted and respond accordingly in the event of an abnormality.

At the preparation phase of an attack, the attacker infects the slave computers and uses these compromised hosts to gather security related information from the target system [6]. This phase is generally called as Command and Control. We assume that the attackers scan the target systems for vulnerabilities before the actual attack occurs. Our system will also log these scanning events and abnormal behaviors. Thus, we will have the information of suspiciously infected regions and the malicious requests which are generated from those regions. With the help of the actual normal behavior knowledge base, we will analyze and log the abnormalities by region and time-stamps. So, in the case of an attack, we can cease the connection between the target system and the regions which are marked as suspicious in the recent log history. Command and Control (C&C) Server, uses their bots to scan the target system, and

they generally attempt to mimic normal traffic to avoid to be detected [32]. There are many different C&C techniques to establish communication and gather information from target system. We have listed some of these techniques which can be detected by our proposed system:

- **Application Layer Protocol Attacks:** Commands which will be sent to the target system is usually disguised in common application layer protocols such as HTTP(S), DNS or SMTP.
- **Non-Application Layer Protocol Attacks:** Internet Control Message Protocol (ICMP), User Datagram Protocol (UDP) are a few of many examples of these protocols.
- **Attacks to Common Port(s)** : Attackers can communicate via commonly used ports such as HTTP:80, DNS:53, SMTP:25, to blend with benign traffic to stay undetected by firewalls or IDS infrastructures.
- **Attacks to Uncommon Port(s):** Attackers might also use non-standard ports to bypass misconfigured firewalls.
- **Custom C&C Protocol Attacks:** This is a non-traditional but a robust way to bypass intrusion detection systems. Attackers might introduce a custom protocol on top of known application layer protocols by imitating well-known protocols.

To be able to test our prototype for the aforementioned attack techniques, we are going to analyze the network traffic data of an exemplary client system, and with the help of our proposed machine learning based model, our intention is to alert the system administrators for abnormal traffic observations beforehand, or help them to be able to cease the connection to the infected regions rather than the legitimate ones during an attack.

As aforementioned before, our proposed IDS, will learn from the previously seen traffic data and it will give alarms if any abnormal behaviour is observed. To automate this functionality, our system will utilize from machine learning techniques. In machine learning, there are three types of learning strategies; unsupervised learning, supervised learning and semi-supervised learning.

In *supervised learning*, the training data is labelled with normal and abnormal behaviours; so the algorithm which is applied on the data, learns with the help of those labels and stops when the required precision is acquired. On the contrary, in unsupervised learning, the data is not labelled and there is no prior knowledge about which part of the data might be abnormal. The aim in *unsupervised learning* is to generate a distribution for normal behaviour and discovering the abnormal behaviours. Unsupervised learning can be used for outlier (novelty) detection, pattern recognition and data analysis. In *semi-supervised learning*, the data is partially labelled and the output for every input is not properly defined.

In this study, even though we have a labelled cybersecurity dataset, we are not going to use supervised learning, because we want our system to be able to change strategies according to the incoming real-time traffic data. Because in reality, we are not able to precisely know which flow is benign or botnet-generated. Therefore, we will treat our data as unlabeled and apply unsupervised learning algorithms.

3.3 Data Preparation and Transformation

In machine learning, data preparation is a very important process, since it has a huge impact on the performance of the models. We are going to slightly manipulate our data, since there are some unnecessary fields which we do not want our model to consider in training and also we will calculate some more additional features which we have aforementioned in Section 2.5.1.

3.3.1 Data manipulation

First of all, in order to prepare the training and testing data to match up with our scenario, we needed to add regions to each individual flow, according to the source hosts' IP addresses. To satisfy this requirement, we have determined a 6-region set which are numbered 1 to 6 as R1, R2...R6. The regions are divided into two classes specified as Infected (R_i) and Benign (R_b).

$$R_{i,b} = \{i,b \mid 1 \leq i,b \leq 6 \text{ and } 3 \leq i \leq 5, b \neq i\} \tag{3.1}$$

Since our dataset is labelled with Background, Normal and Botnet labels, we have the information which of the host IP's should be placed in the Infected regions in advance.

Even though we will not use those labels in the training phase, we are going to benefit from those labels in the data preparation phase. We statically assigned infected hosts to one of the infected regions. Table 3.2 gives the assigned regions to the infected hosts.

Table 3.2 : List of infected hosts with their assigned regions.

Infected Host IP	Assigned Region
147.32.84.165	R3
147.32.84.191	R3
147.32.84.192	R3
147.32.84.193	R4
147.32.84.204	R4
147.32.84.205	R4
147.32.84.206	R5
147.32.84.207	R5
147.32.84.208	R5
147.32.84.209	R5

The remainder of the flows are assigned in one of the Benign regions randomly. While assigning those regions, a dictionary of previously assigned regions are hold in the memory; in order to prevent assigning same hosts to different regions.

After assigning the hosts to their regions, we need to clean our data in order to have the best performance from our training phase. When we analyze the dataset, we observe some alphanumeric values for source/destination ports and source/destination TOS bytes, which are not understood as an acceptable machine-readable value for our models. So we need to label encode these fields. As it is generally the case in real life, our data has some missing values which makes it incomplete. Handling missing values is an important task before we train our models. Figure 3.1 shows the missing value percentages of our dataset. As it can be seen from the figure, the biggest percentage of missing value is dTOS field with 0.14 of the overall data. We observed that these data is mainly Background benign data, however it might affect the training phase to lose that much benign data if we drop the rows with missing fields.

We might take multiple approaches to handle with missing data. For example, if the missing values do not constitute a big percentage of the whole dataset, then we might think to drop them [33]. However, this is not a preferable way, since we might be loosing some of important inputs. Instead of that we want to keep them in our dataset by replacing fields with some constant values as shown in listing 3.2, which we are

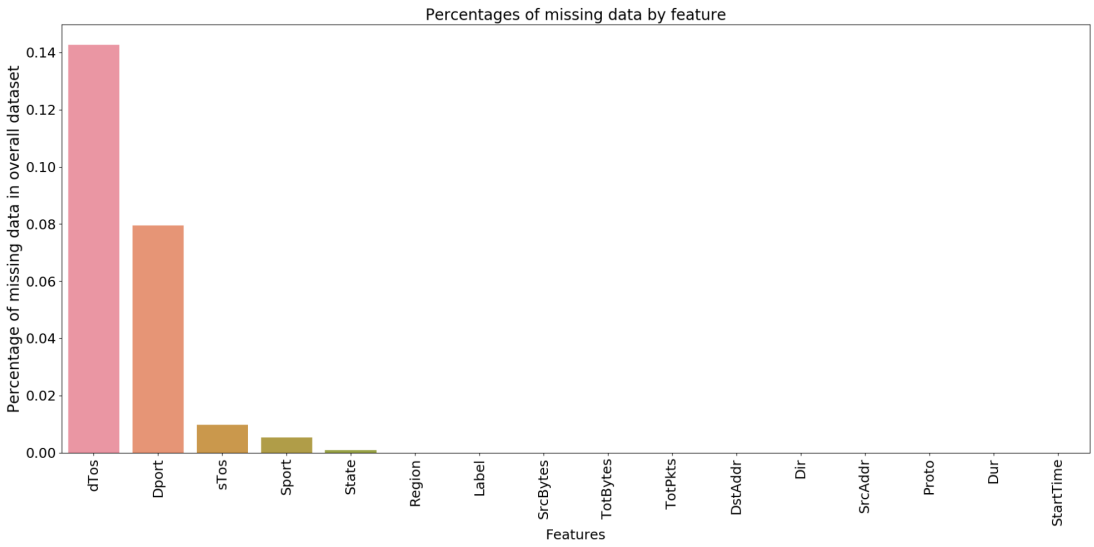


Figure 3.1 : Percentages of missing values in CTU-13 dataset.

sure they are not significant for the corresponding fields or they are the default values. These values are listed below:

- State | EMPTY
- Sport | -99
- Dport | -99
- sTos | 0
- dTos | 0

Numpy understands those values as NaN (not a number). So we have to drop these individual rows for the sake of our model.

```
import pandas as pd

data = pd.read_csv("regional_data.csv")
data = data.drop(['StartTime'], axis=1)
data = data.drop(['Label'], axis=1)
data.State.fillna('EMPTY', inplace=True)
data.Sport.fillna('-99', inplace=True)
data.Dport.fillna('-99', inplace=True)
data.sTos.fillna('0', inplace=True)
data.dTos.fillna('0', inplace=True)
```

Figure 3.2 : Filling NaN rows and dropping unused columns with pandas.

We also dropped some of the columns, which we do not utilize from and which we are not able to encode. The dropped columns are StartTime, hold as a long datetime

Table 3.3 : List of CTU-13 Scenario 10 dataset columns.

Column Name	Description
StartTime	Start time of the flow.
Dur	Flow duration.
Proto	Protocol type.
SrcAddr	Source IP address.
Sport	Source port.
Dir	Direction of the flow.
DstAddr	Destination IP Address.
Dport	Destination Port.
State	State.
sTos	Source TOS byte value.
dTos	Destination TOS byte value.
TotPkts	Total packets.
TotBytes	Total bytes.
SrcBytes	Source bytes.
Label	Label (Background, Normal or Botnet).

format, and Label column, which indicates if the flow is benign or not. Label columns is not necessary since we are going to try unsupervised learning algorithms and instead of StartTime we are going to calculate some new features related to interarrival times as we defined in 2.5.1.

3.3.2 Feature selection

Feature selection is the key step of the data preparation process, because if the features are not selected wisely, overfitting or decreased accuracy might be a few of many problems that we might face.

Table 3.3 gives the list of the columns in Scenario 10 data. For effective anomaly detection, we will calculate a few additional features and use these features to train our model.

For each host in the dataset, the additional features which are defined in Section 2.5.1.

3.3.3 Label encoding

Label encoding is a transformation technique which normalizes and turns them into machine-readable numeric labels. In machine learning, there is no way to use string values to run models on. This technique can only be applied when there is a finite number of classes on the corresponding label. Otherwise, models will throw "y

contains previously unseen labels" error on transformation process. In our case, we need to transform some string values to their numerical representations. For this purpose, we used *LabelEncoder* utility class of sklearn library under *preprocessing* namespace. We encoded SrcAddr, Proto, DstAddr, Sport, Dport, Dir and State fields. An example encode-decode transformation in sklearn can be seen in Figure 3.3.

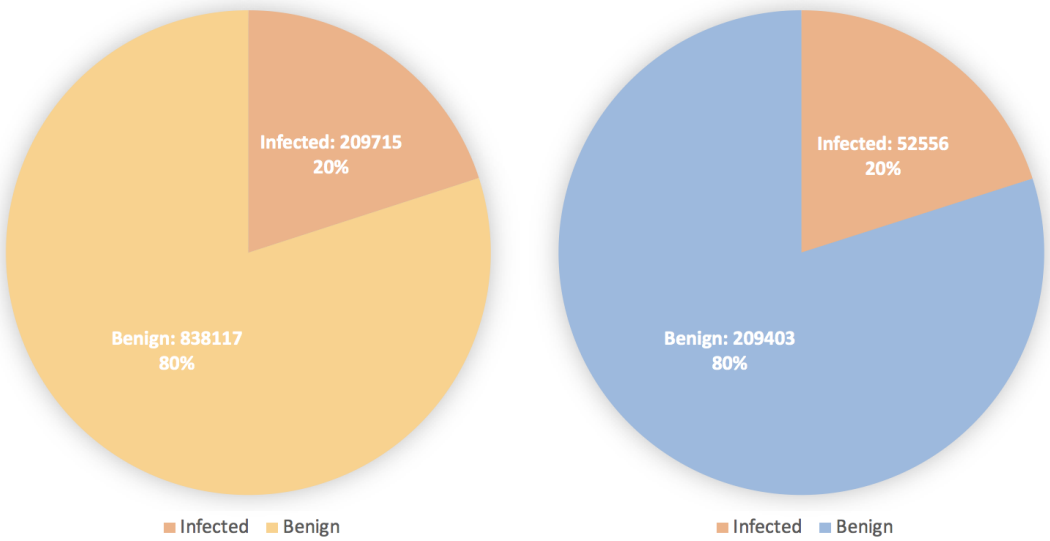
```
from sklearn.preprocessing import LabelEncoder

label_encoder = LabelEncoder()
source_encoder = label_encoder.fit(data.SrcAddr)
data.SrcAddr = source_encoder.transform(data.SrcAddr)
print(data.head(1).SrcAddr) # print the first element in dataframe
>>3
original_srcAddr = source_encode.inverse_transform(data.SrcAddr)
print(data.head(1).SrcAddr)
>>147.168.3.1
```

Figure 3.3 : Transformation and inverse transformation with label encoding.

3.4 Preparing Train and Test Data

In Scenario 10 dataset, there are 1.309.791 total flows and 106.365 of the total flows interfered with infected hosts. Namely, 92% of the whole data set contains benign flows and 8% of the flows consists botnets. When splitting our data to train and test sets, we might take the same approach as stated in [16], which they made sure to satisfy 1:10 ratio for botnet:benign data. However, even this approach will lead us to have a correct reflection of the real-data set, it might also lead us to overfitting. We will use train_test_split method in Scikit-learn library.



(a) Training set (b) Testing set

Figure 3.4 : Contamination ratios of testing and training sets.

For the sake of consistency, we created one training and one testing dataset and we used these sets for all of our experiments. We used 8:2 ratio for training and testing sets. And we used random number generators to select random rows to put into each set, since we want random contamination in each set. Coincidentally, random number generator, generated our data with 2:8 malicious:benign data. Figure 3.4 shows the numbers of benign and infected flows for test and train sets. Training dataset has total 1047832 flows, which 0.20014 of it is infected flows; whereas testing dataset has total 261959 flows, which 0.20062 of it is infected flows.

4. RESULTS

We have tried three different unsupervised learning algorithms for our dataset. At first, we tried to train the model with the default features. The accuracy values of the three classifiers ranged from 0.73 to 0.76 as shown in Table 4.1. The results are not surprising, since there is only a limited malicious attack traffic. A naive prediction algorithm could only detect 10% of the real contamination. As seen in Table 4.1, there are two metrics that are calculated for each classifier. Recall can be described as the sensitivity value of the classifier, which is calculated as in Formula **(4.1)**.

$$r = \frac{TP}{TP + FN} \tag{4.1}$$

TP indicates the true positives and FN indicates the false negatives. The more the recall value is closer to 1, the better our classifier is able to find all true positives. Accuracy is calculated by comparing the predicted values to the correct labels. Both of the values are calculated with Scikit-learn and validated by manual calculations.

Table 4.1 : Outlier detection results with default features.

	ISOF	LOF	EE
Accuracy	0.7307	0.7566	0.7620
Recall	0.8571	0.9105	0.9140

The three algorithms do not have so sharp differences in terms of accuracy. However, if we want to interpret those values, by looking at the accuracy scores, the Isolation Forest classifier performed the worst, which might be an indicator of that our data cannot handle more feature dimensions. We see that the best performing classifier is Elliptic Envelope, which indicates that our inliers might have a Gaussian distribution. If the inliers were less unimodal, EE is expected to degrade in the overall classification performance.

In the upcoming sections, we will share the classifier performance results when we add additional features, and see whether our classifiers performs better with detecting outliers with newly introduced features or not.

4.1 Number of Different Protocol Types

The number of different protocols that one source IP is using is an interesting metric to understand the behaviors of intruders. Because, usually 1 or 2 types of protocols are used by one source. If there are multiple various transmission protocols are used, then this might be a sign to an intruder is trying to intrude to the system. The results of classifiers when we add distinct number of protocol types for each flow is shown in Table 4.2.

Table 4.2 : Outlier detection results with protocol type count.

	ISOF	LOF	EE
Accuracy	0.7114	0.7430	0.7675
Recall	0.8450	0.9019	0.9174

When we analyze the results of the three algorithms, there is not much change is seen in accuracies. However, in average, we can say that slightly decrease is seen in accuracies. This is because, in the dataset we use for outlier detection, distinct protocol type count is not a definitive parameter, since the malicious and benign flows usually have the similar number of protocol types. For our dataset average distinct protocol count for malicious traffic is 3.0, and average distinct protocol count for benign traffic is 2.83. Eventually, this metric does not make a huge difference in detecting outliers, even instead, it results a decrease in the accuracies. However, it does not mean that in a generic outlier detection system, this metric should not be used. On the contrary, we still believe that, it might be a good indication of malicious traffic is flowing through the target system.

4.2 Average Interarrival Time

Another important metric to catch malicious traffic is the average interarrival time of a request which is coming from the same host in a specific time window. The calculated results are promising, adding slightly more accuracy to each of the three classifier results which can be seen in Table 4.3.

Table 4.3 : Outlier detection results with interarival time.

	ISOF	LOF	EE
Accuracy	0.73694	0.7537	0.7458
Recall	0.8594	0.9123	0.9041

This value is the first time-related feature used in our training phase since at the beginning we had opt out the StartTime field. From the results, it can be seen that the accuracy values for ISOF and LOF algorithms are slightly increased, however with the new feature, EE did not perform well and we observe a decreased accuracy results with EE classifier. This might be interpreted as that EE classifier might not perform well with time-based features.

4.3 Average Packet Length

This parameter is the average packet size of individual host sends. In legitimate traffic, this value is typically larger, and for malicious trafic, the packet sizes are relatively small. When we analyze the distribution of packet sizes on time domain, we can easily see this trend. In benign traffic 4.1 trends, the packet sizes varies in length. However, in malicious traffic trend 4.2, if we disregard the exceptional peeks, the sizes of packets are at an ordinary, persistent level.

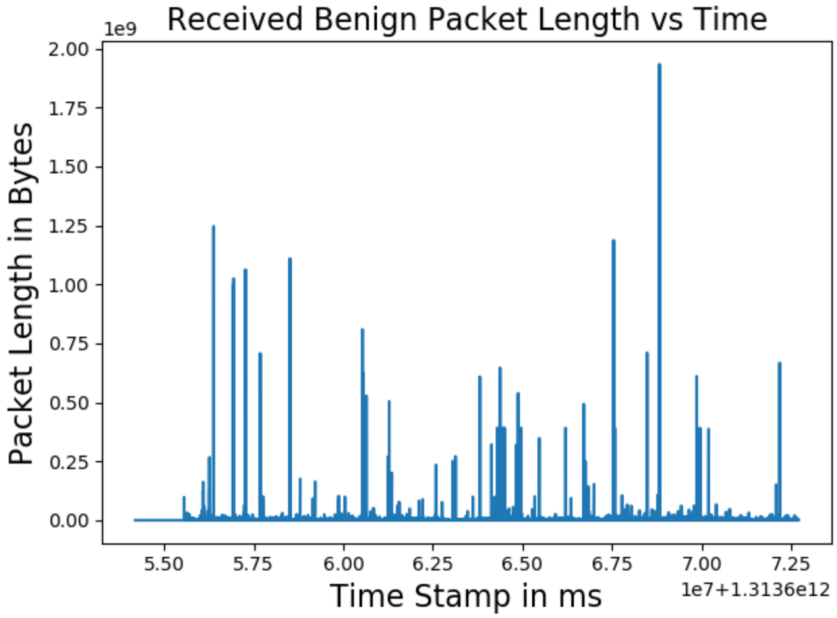


Figure 4.1 : Distribution of benign packet sizes on time domain.

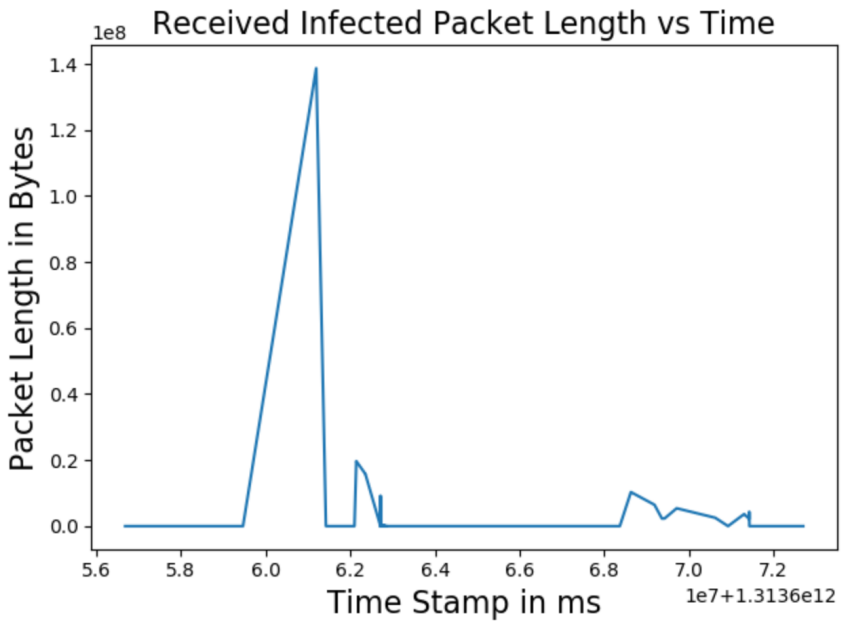


Figure 4.2 : Distribution of malicious packet sizes on time domain.

After calculation of average packet length values in one time window for each specific host, we trained our model again. However, we did not reach a good accuracy since this value is not so different than individual packet sizes which are already in the default feature set. So, we did not share the results here.

4.4 Same-length number of packets ratio

This is another interesting parameter to consider while training our model, because for legitimate flows, this value is expected to be small. As it can be seen from 4.2, malicious traffic has a trend to have the similar-length packets over time domain. Therefore, when you proportion same-length packet count over total packet count in a time window, especially for legitimate flows this value is expected to close to zero.

Additional to interarrival-time, this metric helps us to increase the accuracy for our selected classifiers as seen in Table 4.4.

Table 4.4 : Outlier detection results with same-length packet ratio.

	ISOF	LOF	EE
Accuracy	0.76694	0.7837	0.7558
Recall	0.8794	0.9223	0.9141

5. OPEN ISSUES & CHALLENGES & FUTURE WORK

Machine learning-based systems have the advantages of being responsive to changes and being able to adapt their strategy according to incoming new data. In that sense, our system is capable of detecting unprecedented botnet attacks with the help of its continuously-learning nature. Nevertheless, machine learning-based systems have some drawbacks. One major flaw in those systems is that they are not efficient in terms of resource consumption. In that sense, our model is also resource-expensive.

Our system encounters all of the disadvantages which a distributed IDS faces. For example, it is hard to maintain a distributed system, because it has much more components which should be kept running, compared to the centralized systems. Recovery and fault tolerance is harder in our systems, because it is not easy to hold the current state in a consistent structure because of scattered behavior of whole system.

Intrusion detection systems are set up as precautions to misuse. They are more of a detect-and-report systems, rather than being preventive systems [34] against attacks. In our design, we are currently writing the results to ELK Stack, however, as a future work, it is possible to add some preventive modules to our open-source architecture, which will detect attacks and cease connection where necessary.

Another aspect which can counted as a drawback is that we only made predictions with a specific dataset. As we know that, botnets evolve with time, our system should be aware of different type of attacks.

We are aware of that our POC does not necessarily mean that in every data set we will get the similar precisions, since the attacks evolve and botnet behaviors change in time.

6. CONCLUSIONS

In this work, in an effort to detect or predict the locations of slave computers, we proposed a novel, distributed, scalable, machine-learning based Intrusion Detection System (IDS). Our system can handle and process huge volume of data with low-latency. The Machine Learning Engine (ML Engine) in our IDS, continuously learns from real-time data and make predictions accordingly. We would like our work to be seen as a practical examplary ML based IDS. Nevertheless, we are aware of that our POC does not mean that in every data set we will get the similar precisions, since the attacks evolve and botnet behaviors change in time.

REFERENCES

[1] **Url-1**, Documentation of scikit-learn 0.21.3, `https://scikit-learn.org/stable/documentation.html`, date retrieved : 28.04.2019.

[2] **Pedregosa, F.**, **Varoquaux, G.**, **Gramfort, A.**, **Michel, V.**, **Thirion, B.**, **Grisel, O.**, **Blondel, M.**, **Prettenhofer, P.**, **Weiss, R.**, **Dubourg, V.**, **Vanderplas, J.**, **Passos, A.**, **Cournapeau, D.**, **Brucher, M.**, **Perrot, M. and Duchesnay, E.** (2011). Scikit-learn: Machine Learning in Python, *Journal of Machine Learning Research*, *12*, 2825–2830.

[3] **Womersley, R.** (2017). When will exponential mobile growth stop?, *LS Telcom AG*.

[4] **Url-2**, Protection and mitigation techniques using managed Distributed Denial of Service (DDoS) protection service, `https://aws.amazon.com/shield/ddos-attack-protection/`, date retrieved : 20.03.2019.

[5] **Cisco Annual Cybersecurity Report**, (2018), Cisco Systems.

[6] **Bhattacharyya, D.K. and Kalita, J.K.** (2016). *DDoS Attacks Evolution, Detection, Prevention, Reaction, and Tolerance*, CRC Press, 6000 Broken Sound Parkway NW, Suite 300, 1 edition.

[7] **Poongothai, M. and Sathyakala, M.** (2012). Simulation and analysis of DDoS attacks, *2012 International Conference on Emerging Trends in Science, Engineering and Technology (INCOSET)*, pp.78–85.

[8] **Kahn, C.E.**, **Porras, P.A.**, **Staniford-Chen, S. and Tung, B.** (2000). A common intrusion detection framework.

[9] **Buczak, A.L. and Guven, E.** (2016). A Survey of Data Mining and Machine Learning Methods for Cyber Security Intrusion Detection, *IEEE Communications Surveys Tutorials*, *18*(2), 1153–1176.

[10] **Url-3**, What Is Cybersecurity?, `https://www.cisco.com/c/en/us/products/security/what-is-cybersecurity.html`, date retrieved : 28.04.2019.

[11] **García-Teodoro, P.**, **Díaz-Verdejo, J.**, **Maciá-Fernández, G. and Vázquez, E.** (2009). Anomaly-based network intrusion detection: Techniques, systems and challenges, *Computers Security*, *28*(1), 18 – 28.

[12] **Lippmann, R.P. and Cunningham, R.K.** (2000). Improving intrusion detection performance using keyword selection and neural networks, *Computer Networks*, *34*(4), 597 – 603, recent Advances in Intrusion Detection Systems.

[13] **Apiletti, D.**, **Baralis, E.**, **Cerquitelli, T. and D'Elia, V.** (2009). Characterizing network traffic by means of the NetMine framework, *Computer Networks*, *53*(6), 774 – 789, traffic Classification and Its Applications to Modern Networks.

[14] **Borders, K.**, **Xin Zhao and Prakash, A.** (2006). Siren: catching evasive malware, *2006 IEEE Symposium on Security and Privacy (S P'06)*, pp.6 pp.–85.

[15] **Narkhede, N.**, **Shapira, G. and Palino, T.** (2017). *Kafka: The Definitive Guide Real-Time Data and Stream Processing at Scale*, O'Reilly Media, Inc., 1st edition.

[16] **Ding, S.** (2018). Machine Learning for Cybersecurity: Network-based Botnet Detection Using Time-Limited Flows, *Caltech Undergraduate Research Journal*.

[17] **Saad, S.**, **Traore, I.**, **Ghorbani, A.**, **Sayed, B.**, **Zhao, D.**, **Lu, W.**, **Felix, J. and Hakimian, P.** (2011). Detecting P2P botnets through network behavior analysis and machine learning, *2011 Ninth Annual International Conference on Privacy, Security and Trust*, pp.174–180.

[18] **Biglar Beigi, E.**, **Hadian Jazi, H.**, **Stakhanova, N. and Ghorbani, A.A.** (2014). Towards effective feature selection in machine learning-based botnet detection approaches, *2014 IEEE Conference on Communications and Network Security*, pp.247–255.

[19] **Yu, X.**, **Dong, X.**, **Yu, G.**, **Qin, Y. and Yue, D.** (2010). Data-Adaptive Clustering Analysis for Online Botnet Detection, *2010 Third International Joint Conference on Computational Science and Optimization*, volume 1, pp.456–460.

[20] **Livadas, C.**, **Walsh, R.**, **Lapsley, D. and Strayer, W.T.** (2006). Usilng Machine Learning Technliques to Identify Botnet Traffic, *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, pp.967–974.

[21] **Strayer, W.T.**, **Lapsely, D.**, **Walsh, R. and Livadas, C.**, (2008). Botnet Detection Based on Network Behavior, Springer US, Boston, MA, pp.1–24.

[22] **Strayer, W.T.**, **Walsh, R.**, **Livadas, C. and Lapsley, D.** (2006). Detecting Botnets with Tight Command and Control, *Proceedings. 2006 31st IEEE Conference on Local Computer Networks*, pp.195–202.

[23] **Url-4**, Machine Learning: An In-Depth Guide Data Selection, Preparation, and Modeling, `https://medium.com/@innoarchitech/machine-learning-an-in-depth`, date retrieved : 12.02.2019.

[24] **Halevy, A.**, **Norvig, P. and Pereira, F.** (2009). The Unreasonable Effectiveness of Data, *IEEE Intelligent Systems*, *24*, 8–12.

[25] **Url-5**, numpy.random.RandomState, `https://docs.scipy.org/doc/numpy-1.16.1/reference/generated/numpy.random.RandomState.html`, date retrieved: 06.08.2019.

[26] **Rousseeuw, P.J. and Driessen, K.V.** (1999). A Fast Algorithm for the Minimum Covariance Determinant Estimator, *Technometrics*, *41*(3), 212–223.

[27] **Hoyle, B.**, **Michael Rau, M.**, **Paech, K.**, **Bonnett, C.**, **Seitz, S. and Weller, J.** (2015). Anomaly detection for machine learning redshifts applied to SDSS galaxies, *Monthly Notices of the Royal Astronomical Society*, *452*.

[28] **Url-6**, (2018), Introduction to Kafka Connectors, `https://www.baeldung.com/kafka-connectors-guide`, date retrieved : 12.08.2019.

[29] **Url-7**, Kafka Connect Elasticsearch Sink Connector, `https://docs.confluent.io/current/connect/`.

[30] **Url-8**, (2017), Kafka Connect Elasticsearch: Consuming and Indexing with Kafka Connect, `https://sematext.com/blog/kafka-connect-elasticsearch-how-to/`, date retrieved: 06.08.2019.

[31] **Tariq, F. and Baig, S.** (2017). Machine Learning Based Botnet Detection in Software Defined Networks, *International Journal of Security and Its Applications*, *11*, 1–12.

[32] **Url-9**, Command and Control, `https://attack.mitre.org/tactics/TA0011/`, date retrieved: 06.08.2019.

[33] **Url-10**, Handling Missing Values in Machine Learning: Part 1, `https://towardsdatascience.com/dda69d4f88ca`, date retrieved: 06.08.2019.

[34] **Spafford, E.H. and Zamboni, D.** (2000). Intrusion Detection Using Autonomous Agents, *Comput. Netw.*, *34*(4), 547–570.

CURRICULUM VITAE

Name Surname : Zemre ARSLAN TÜVER

Place and Date of Birth : İstanbul, 08.08.1990

E-Mail : zemre.arslan@itu.edu.tr

EDUCATION :

- **B.Sc.** : 2013, Istanbul Technical University, Computer and Informatics Faculty, Computer Engineering

OTHER PUBLICATIONS, PRESENTATIONS AND PATENTS:

- M. Erel, **Z. Arslan**, Y. Ozcevik, B. Canberk, 2014. Software-Defined Wireless Networking: A New Paradigm for Next Generation Network Management Framework, *Modelling and Simulation of Computer Networks and Systems: Methodologies and Applications,* Edited by M.S. Obaidat, F. Zarai and P. Nicopolitidis, Elsevier Publications.

- M. Erel, **Z. Arslan**, Y. Ozcevik, and B. Canberk, 2014. Grade of Service (GoS) based Adaptive Flow Management for Software Defined Heterogeneous Networks (SDHetN), *Computer Networks (Elsevier),* DOI: 10.1016/j.comnet.2014.11.012.

- **Z. Arslan**, M. Erel, Y. Ozcevik and B. Canberk, 2014. Sdoff: A Software-Defined Offloading Controller for Heterogeneous Networks, *IEEE Wireless Communications and Networking Conference, IEEE WCNC,Istanbul-Turkey,* April 2014.

- **Z. Arslan**, A. Alemdaroglu and B. Canberk, 2013. A Traffic-Aware Controller Design for Next-Generation Software Defined Networks, *IEEE International Black Sea Conference on Communications and Networking, IEEE BLACKSEACOM, Batumi-Georgia,* June 2013.