



Faculty of Engineering

Computer and Systems Engineering Department

Computer Engineering and Software Systems Program

Graduation Project

Big-Data Analytics-Based IPS System

Submitted by

Abdullah Zakaria Mohamed

Yasmin Shaaban Mahmoud

Mohsen Ahmed Mohsen

Emad Adel Ali El-Din

Supervisor

Dr. Gamal Abdel Shafy Ebrahim

2017

Table of Contents

List of Figures	4
List of Tables	6
Abstract:.....	9
1 Introduction	10
2 Intrusion prevention system approaches	14
3 About project	16
3.1 Problem Statement	16
3.2 Project Goals and Deliverables	17
4 System Analysis and Design	18
4.1 Software Development Life Cycle	18
4.2 System Analysis	22
4.2.1 System Requirements	22
4.2.2 System use case diagram	23
4.2.3 Proposed Architecture Model.....	25
4.3 System Design	26
4.3.1 System Class diagram.....	26
4.3.2 System Sequence diagram	28
5 Project Management.....	29
5.1 Used Tools	29
5.2 Teamwork.....	29
5.3Planning	30
6 Project Data Analytics Lifecycle.....	32
6.1 Discovery	33
6.1.1 Search about Intrusion Prevention System.....	33
6.1.2 Learn from the past.....	33
6.1.3 Resources	34
6.1.4 Frame the problem	34
6.1.5 Formulation of Initial Hypothesis.....	35
6.2 Data Preparation	36
6.2.1 Capturing packet	36
6.2.2 Splitting packet fields	36
6.2.3 Creating datasets	57
6.2.4 Analyzing Snort Signature based system	71

6.3 Model Planning	73
6.3.1 Techniques and Workflows and Model Selection.....	73
6.4 Model Building	74
6.4.1 Categorization	74
6.4.2 Text analysis techniques	74
6.4.3 Classification.....	75
6.4.4 Accessing firewall and adding needed rules	84
6.5 Testing Tool Cloud Community Data Bricks	94
6.6 Operationalize	99
7 User Guide.....	100
8 Output Results and Future Development	104
8.1 Overview	104
8.2 Snippets of output.....	105
8.3 Project Future Development.....	108
References	109

List of Figures

Figure 1 Comparison between IPS and IDS [1].....	11
Figure 2 Application layer in network reference model	14
Figure 3 Transport and Network layers in network reference model	15
Figure 4 Software Development Life Cycle Activities [2]	18
Figure 5 Use case Diagram	23
Figure 6 Proposed Repository Architecture Model.....	25
Figure 7 Class Diagram	26
Figure 8 Sequence Diagram	28
Figure 9 Gantt chart Part 1.....	30
Figure 10 Gantt chart Part 2.....	30
Figure 11 Gantt chart Part 3.....	31
Figure 12 Gantt chart Part 4.....	31
Figure 13 Data Analytics Lifecycle	32
Figure 14 IP header [3]	37
Figure 15 TCP header [4]	44
Figure 16 UDP header [5]	48
Figure 17 ICMP header [6]	49
Figure 18 Snort rule form [7].....	71
Figure 19 Example of Training Data and Decision tree model.....	77
Figure 20 OFDS Decision tree model.....	79
Figure 21 Gaussian curve [8]	80
Figure 22 Example on OFDS Decision tree model.....	83
Figure 23 Accessing Netsh AdvFirewall Firewall	93
Figure 24 Example clusters [10]	95
Figure 25 Figure 25 Partitioning data frame to “Resilient Distributed Datasets” (RDDs) [10] ...	96
Figure 26 Spark scalability.....	99
Figure 27 two windows	100
Figure 28 path example.....	100
Figure 29 IPSMainReceiver.....	101
Figure 30 IPSMainSender	101
Figure 31 IPSMainSender running.....	102

Figure 32 IPSMainReceiver running	102
Figure 33 when pressing CTRL+C shows message System is terminated	103
Figure 34 when typing python Firewall.py.....	103
Figure 35 Example of sniffing packet and analyzing IP Header and TCP Header.....	105
Figure 36 Example of sniffing packet and analyzing IP Header and UDP Header.....	106
Figure 37 Example of sniffing packet and analyzing IP Header and ICMP Header	107

List of Tables

Table 1 test outcomes	35
Table 2 Version field in IP header	37
Table 3 Version bits and Description	37
Table 4 Internet Header Length field in IP header	38
Table 5 Differentiated Services field in IP header	38
Table 6 Differentiated Services bits	38
Table 7 Type of Service Description	38
Table 8 Type of Service bits.....	39
Table 9 Precedence field values	39
Table 10 D bit values	39
Table 11 T bit values.....	40
Table 12 R bit values	40
Table 13 M bit values	40
Table 14 Total length field in IP header	40
Table 15 Identification field in IP header	41
Table 16 Flags bits and its description	41
Table 17 Don't fragment bit values.....	42
Table 18 More fragments bit values	42
Table 19 Fragment Offset field in IP header	42
Table 20 Time to live field in IP header	42
Table 21 Protocol field in IP header	43
Table 22 Protocol field Values analyzed in project	43
Table 23 Header checksum field in IP header	43
Table 24 Source IP address field in IP header	43
Table 25 Destination IP address field in IP header.....	43
Table 26 Source Port field in TCP header	44
Table 27 Destination Port field in TCP header	44
Table 28 Sequence Number field in TCP header	45
Table 29 Acknowledgment Number field in TCP header	45
Table 30 Data Offset field in TCP header	45
Table 31 Reserved field in TCP header.....	46
Table 32 ECN, Explicit Congestion Notification bits	46
Table 33 Control bits	46

Table 34 Window field in TCP header	47
Table 35 Checksum field in TCP header	47
Table 36 Urgent Pointer field in TCP header.....	47
Table 37 Source Port field in UDP header	48
Table 38 Destination Port field in UDP header	48
Table 39 Length field in UDP header.....	48
Table 40 Checksum field in UDP header	49
Table 41 Type field in ICMP header	49
Table 42 Type field values [11].....	50
Table 43 Code field in ICMP header	52
Table 44 Type 0 in code field.....	52
Table 45 Type 3 in code field.....	52
Table 46 Type 5 in code field.....	53
Table 47 Type 8 in code field.....	54
Table 48 Type 9 in code field.....	54
Table 49 Type 10 in code field	54
Table 50 Type 11 in code field	54
Table 51 Type 12 in code field	55
Table 52 Type 13 in code field	55
Table 53 Type 14 in code field	55
Table 54 Type 40 in code field	56
Table 55 ICMP Header Checksum field	56
Table 56 TCP Dataset Part 1	57
Table 57 TCP Dataset Part 2	58
Table 58 TCP Dataset Part 3	59
Table 59 TCP Dataset Part 4.....	59
Table 60 TCP Dataset Part 5	59
Table 61 TCP Dataset Part 6	60
Table 62 UDP Dataset Part 1	64
Table 63 UDP Dataset Part 2	65
Table 64 UDP Dataset Part 3	65
Table 65 UDP Dataset Part 4	66
Table 66 UDP Dataset Part 5	66
Table 67 ICMP Dataset Part 1	67

Table 68 ICMP Dataset Part 2	67
Table 69 ICMP Dataset Part 3	68
Table 70 ICMP Dataset Part 4	68
Table 71 any protocol Dataset Part 1.....	69
Table 72 any protocol Dataset Part 2.....	69
Table 73 any protocol Dataset Part 3.....	70
Table 74 any protocol Dataset Part 4.....	70

Abstract:

Intrusion detection is so much popular since the last two decades where intrusion is attempted to break into or misuse the system. It is mainly of two types based on the intrusions, first is Misuse or signature based detection and the other is Anomaly detection. Recent growth of the Internet has been phenomenal and consequently, the computers and the networks that make the Internet hum have become the targets of enemies and criminals. Intrusions into a computer or network system are activities that destabilize them by compromising security in terms of confidentiality, availability or integrity, the three main characteristics of a secure and stable system. Intrusion Prevention is preventing intrusions after detecting them using intrusion detection. We use big data analytics techniques in detecting intrusions then accessing firewall to prevent them. Our main concern is analyzing network traffic and studying user applications behavior to detect malicious behavior.

1 Introduction

In “**Big-Data Analytics-Based IPS System**” project, we exploit predictive analytics to derive a predictive model for the Intrusion Prevention System (IPS) in networking environments. This model will find relations between usage scenarios and how they can be classified as intrusion activities.

Let's talk first about the difference between an IDS and an IPS, let's just lump the two of them together and discuss how they are different than firewalls.

Think of the firewall as the person looking at your boarding pass and ID at the airport before you get to your gate. He's checking what is really the equivalent of the source IP address → who you are, where you live.

The destination IP address → where you're flying to.

The protocol → your airline.

The port → your flight number.

The IDS's or IPS's would be the security guards on the other side by the gates where the people wait to board their planes. A passenger who got by the initial screening might start causing problems by the gate, possibly getting loud and violent over the delay of a flight. He might even be entering areas restricted for airport employees. The firewall, the TSA screener who looked at your boarding pass, can't help at this point.

The problem exists beyond that location now.

The IDS or IPS kicks in, which requires more logic and learning. IDS's and IPS's have to make decisions on where certain lines were crossed and then take appropriate action. The firewall, the TSA agent looking at your board pass, is still a very necessary component. If everyone was just let through to the gates, the airport guards would be overwhelmed and wouldn't be able to monitor all potential passengers. The firewall weeds out those that shouldn't go in, but the IDS or IPS adds a new dimension for those passengers that made it past the first screening.

An IDS -- intrusion detection system → is out of band, and simply gets copies of network traffic. It can be as simple as a system getting copies of traffic to inspect through a switch configured to send all traffic to the IDS. The IPS -- intrusion prevention system → is in-line, so original traffic must pass through the IPS. Since the IDS is out-of-band, it doesn't add any latency. An IPS adds latency, since traffic is processed live. If the IDS sensor goes down, possibly after a target attack, traffic will still flow. If an IPS is targeted, attacked, brought down, traffic might stop right there.

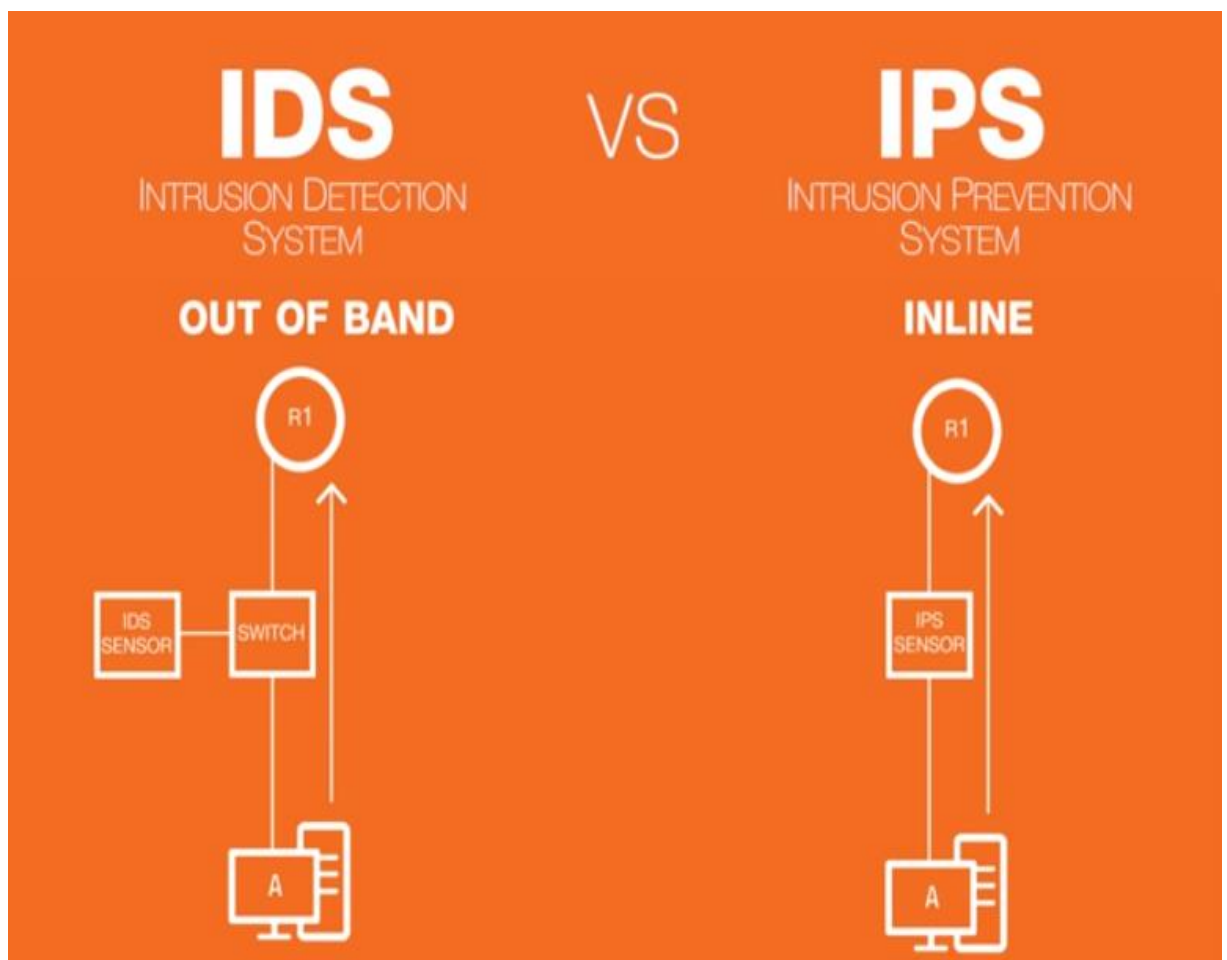


Figure 1 Comparison between IPS and IDS [1]

An Intrusion Prevention System (IPS) is a network security/threat prevention technology that examines network traffic flows to detect and prevent vulnerability exploits. Vulnerability exploits usually come in the form of malicious inputs to a target application or service that attackers use to interrupt and gain control of an application or machine. Following a successful exploit, the attacker can disable the target application (resulting in a denial-of-service state), or can potentially access to all the rights and permissions available to the compromised application.

The IPS often sits directly behind the firewall and it provides a complementary layer of analysis that negatively selects for dangerous content. Unlike its predecessor the Intrusion Detection System (IDS)—which is a passive system that scans traffic and reports back on threats—the IPS is placed inline (in the direct communication path between source and destination), actively analyzing and taking automated actions on all traffic flows that enter the network. Specifically, these actions include:

- Sending an alarm to the administrator (as would be seen in an IDS)
- Dropping the malicious packets
- Blocking traffic from the source address
- Resetting the connection

As an inline security component, the IPS must work efficiently to avoid degrading network performance. It must also work fast because exploits can happen in near real-time. The IPS must also detect and respond accurately, so as to eliminate threats and false positives (legitimate packets misread as threats).

The IPS has a number of detection methods for finding exploits, but signature-based detection and anomaly-based detection are the two dominant mechanisms.

Signature-based detection is based on a dictionary of uniquely identifiable patterns (or signatures) in the code of each exploit. As an exploit is discovered, its signature is recorded and stored in a continuously growing dictionary of signatures. Signature detection for IPS breaks down into two types:

- **Exploit-facing** signatures identify individual exploits by triggering on the unique patterns of a particular exploit attempt. The IPS can identify specific exploits by finding a match with an exploit-facing signature in the traffic stream
- **Vulnerability-facing** signatures are broader signatures that target the underlying vulnerability in the system that is being targeted. These signatures allow networks to be protected from variants of an exploit that may not have been directly observed in the wild, but also raise the risk of false-positives.

Anomaly detection takes samples of network traffic at random and compares them to a pre-calculated baseline performance level. When the sample of network traffic activity is outside the parameters of baseline performance, the IPS takes action to handle the situation.

IPS was originally built and released as a standalone device in the mid-2000s. This however, was in the advent of today's implementations, which are now commonly integrated into Unified Threat Management (UTM) solutions (for small and medium size companies) and next-generation firewalls (at the enterprise level).

2 Intrusion prevention system approaches

First, using signatures known worldwide by security experts to detect malicious network traffic attacks previously known.

Signatures are caught by studying application behavior in stable state. When reported that there is an attack or unexpected behavior for an app using internet. Security experts detect the changes in packets that hold data sent and received by app. these changes are called Signature. So, if any packet holds data containing the known signature then it's a defected packet.

In this part we are concerned with application layer in network reference model which deals with packet payload. It's is important to know that signatures are case sensitive sometimes and must be added carefully. Because, you can choose a signature and this is a part of predefined signature. In this case, you will face incorrect classification for malicious packets. So, signatures must be added by an expert. Experts take in view the predefined signatures and there classes.

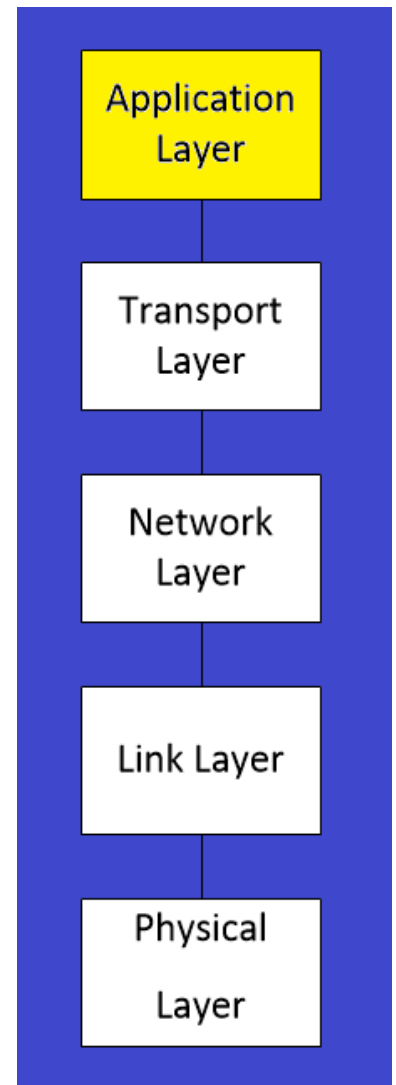


Figure 2 Application layer in network reference model

Second, anomaly based analysis. It's more hard and efficient. It depends on understanding the packet field's content. So, you can put your regular expression model that must match the packet field's content. If there is no match don't just drop the packet. But, put machine learning model that can detect if that behavior of traffic packets is malicious or not "legitimate behavior or not ".

In this part we are concerned with analyzing two layers: network layer and transport layer. Where we analyze the IP header in network layer and analyze transport layer header according to network layer protocol field. We are following the network standard such as RFC for IETF and IANA in detecting anomalies.

We have a famous phrase "abnormal is not normal" while the trend is different "normal is not abnormal". So, what is the difference?!

Our phrase aims to discover abnormal behavior from normal behavior. Security trend aims to discover normal behavior from abnormal behavior.

So, which is better? Ok, answer is pretty easy. It's the one with less time, space and network complexity.

Analyzing normality or abnormality will give same result as answer at the end is yes or no. So, according to time needed to analyze the normality behavior it would take more time in early nineties to

analyze the normality behavior as abnormal behavior done by attacker was less. Security trend was better. But, now our trend is better as there are millions of exploits and vulnerabilities and there will be more according to new operating systems, platforms etc. IoT has also a great effect. We will discuss more in model building section.

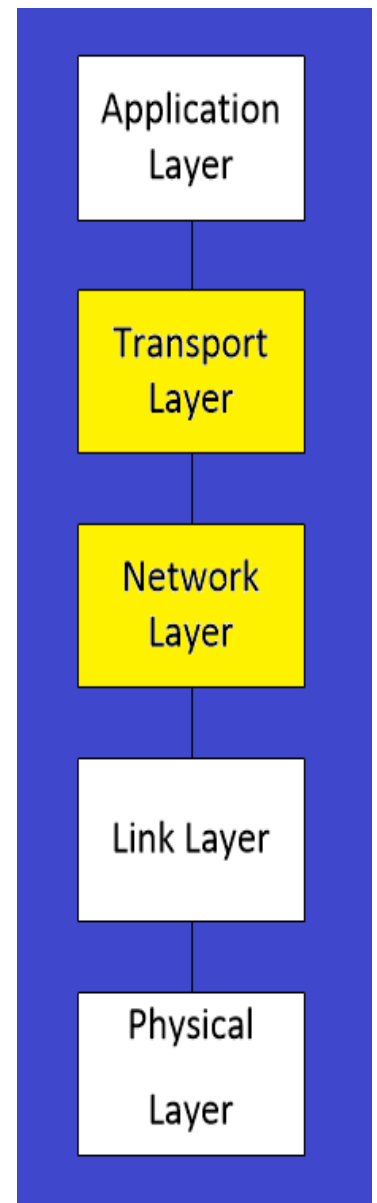


Figure 3 Transport and Network layers in network reference model

3 About project

3.1 Problem Statement

Security is a big issue for all networks in today's enterprise environment. Hackers and intruders have made many successful attempts to bring down high-profile company networks and web services. Using intrusion detection methods, you can collect and use information from known types of attacks and find out if someone is trying to attack your network or particular hosts. The information collected this way can be used to harden your network security and prevent intrusion implementing IPS model.

How about nonprofessional users using internet?

Nowadays, hackers don't attack enterprises by viruses or Trojans to stop their service. They do attack them by their own users and unexpected users.

DDoS (Distributed Denial of Service) hacker attack nonprofessional user. User has low level of security. Hacker adds malicious program in application malware over WWW. User downloads this application without knowing that it contains malware "cracked apps". These applications hackers use them to attack services. Hacker can control app of user and make it request service in no need so computing power of services can't withstand overwhelm request and fail.

3.2 Project Goals and Deliverables

The expected deliverable from this project is an application that can predict/estimate and interact with intruders in computer networks; it will take into consideration the real time nature of the network traffic and the how the prediction models can classify the different activities in a timely manner.

We aim to provide IPS for every PC worldwide as a service in cloud community such as "Amazon EC2, EMC2 Atmos, Microsoft Azure, and Google App Engine". No more DDoS as organizations will support that for nonprofessional users as a service free and paid according to user needs of security. But, organizations are really protecting their services and can use it commercial as advertisement for their cloud service, security as a service.

Our product will be two background service programs: one for listening for incoming connection and other for outgoing connection. These two programs are responsible for creating train data and test data. There is another program on community cloud data bricks which show the power of data analytics platform.

4 System Analysis and Design

4.1 Software Development Life Cycle

Software Development Life Cycle, SDLC for short, is a well-defined, structured sequence of stages in software engineering to develop the intended software product.

SDLC Activities:

SDLC provides a series of steps to be followed to design and develop a software product efficiently. SDLC includes the following steps:



Figure 4 Software Development Life Cycle Activities [2]

Communication

This is the first step where the user initiates the request for a desired software product. He contacts the service provider and tries to negotiate the terms. He submits his request to the service providing organization in writing.

Requirement Gathering

This step onwards the software development team works to carry on the project. The team holds discussions with various stakeholders from problem domain and tries to bring out as much information as possible on their requirements. The requirements are contemplated and segregated into user requirements, system requirements and functional requirements. The requirements are collected using a number of practices as given:

- studying the existing or obsolete system and software.
- conducting interviews of users and developers.
- referring to the database or collecting answers from the questionnaires.

Feasibility Study

After requirement gathering, the team comes up with a rough plan of software process. At this step the team analyzes if a software can be made to fulfill all requirements of the user and if there is any possibility of software being no more useful. It is found out, if the project is financially, practically and technologically feasible for the organization to take up. There are many algorithms available, which help the developers to conclude the feasibility of a software project.

System Analysis

At this step the developers decide a roadmap of their plan and try to bring up the best software model suitable for the project. System analysis includes Understanding of software product limitations, learning system related problems or changes to be done in existing systems beforehand, identifying and addressing the impact of project on organization and personnel etc. The project team analyzes the scope of the project and plans the schedule and resources accordingly.

Software Design

Next step is to bring down whole knowledge of requirements and analysis on the desk and design the software product. The inputs from users and information gathered in requirement gathering phase are the inputs of this step. The output of this step comes in the form of two designs; logical design and physical design. Engineers produce meta-data and data dictionaries, logical diagrams, data-flow diagrams and in some cases pseudo codes.

Coding

This step is also known as programming phase. The implementation of software design starts in terms of writing program code in the suitable programming language and developing error-free executable programs efficiently.

Testing

An estimate says that 50% of whole software development process should be tested. Errors may ruin the software from critical level to its own removal. Software testing is done while coding by the developers and thorough testing is conducted by testing experts at various levels of code such as module testing, program testing, product testing, in-house testing and testing the product at user's end. Early discovery of errors and their remedy is the key to reliable software.

Integration

Software may need to be integrated with the libraries, databases and other program(s). This stage of SDLC is involved in the integration of software with outer world entities.

Implementation

This means installing the software on user machines. At times, software needs post-installation configurations at user end. Software is tested for portability and adaptability and integration related issues are solved during implementation.

Operation and Maintenance

This phase confirms the software operation in terms of more efficiency and less errors. If required, the users are trained on, or aided with the documentation on how to operate the software and how to keep the software operational. The software is maintained timely by updating the code according to the changes taking place in user end environment or technology. This phase may face challenges from hidden bugs and real-world unidentified problems.

Disposition

As time elapses, the software may decline on the performance front. It may go completely obsolete or may need intense upgradation. Hence a pressing need to eliminate a major portion of the system arises. This phase includes archiving data and required software components, closing down the system, planning disposition activity and terminating system at appropriate end-of-system time.

4.2 System Analysis

4.2.1 System Requirements

4.2.1.1 Users, customers and clients

- Ordinary user who wants to protect his/her computer.
- Companies and Enterprises.

4.2.1.2 Functional requirements

- System captures Network traffic, sniff packets and analyze IP, TCP, UDP and ICMP Headers using big data analytics.
- Classifying Packets whether malicious or not using Big Data classification model.
- Detecting attacks and intrusions.
- Showing messages to users with the type of attack.
- Prevent intrusion by dropping malicious packets, blocking network traffic from source address or resetting the connection.

4.2.1.3 Non-Functional requirements

Portability Requirements

- We want to create portable software which specifically designed to run on different computers with compatible operating systems and processors without any machine-dependent installation.

Performance requirements

- Processing the whole data should not exceed 1 minute.
- Accessing data samples should be quick and easy.

Security Requirements

- Review system logs for unauthorized access.
- System should be able to handle any faults made while data processing.
- The data on the system should have a backup in order to prevent complete loss of data.
- System should have a periodic recovery testing. Even if there is no failure happen system should be tested periodically in order to detect any problems.

4.2.2 System use case diagram

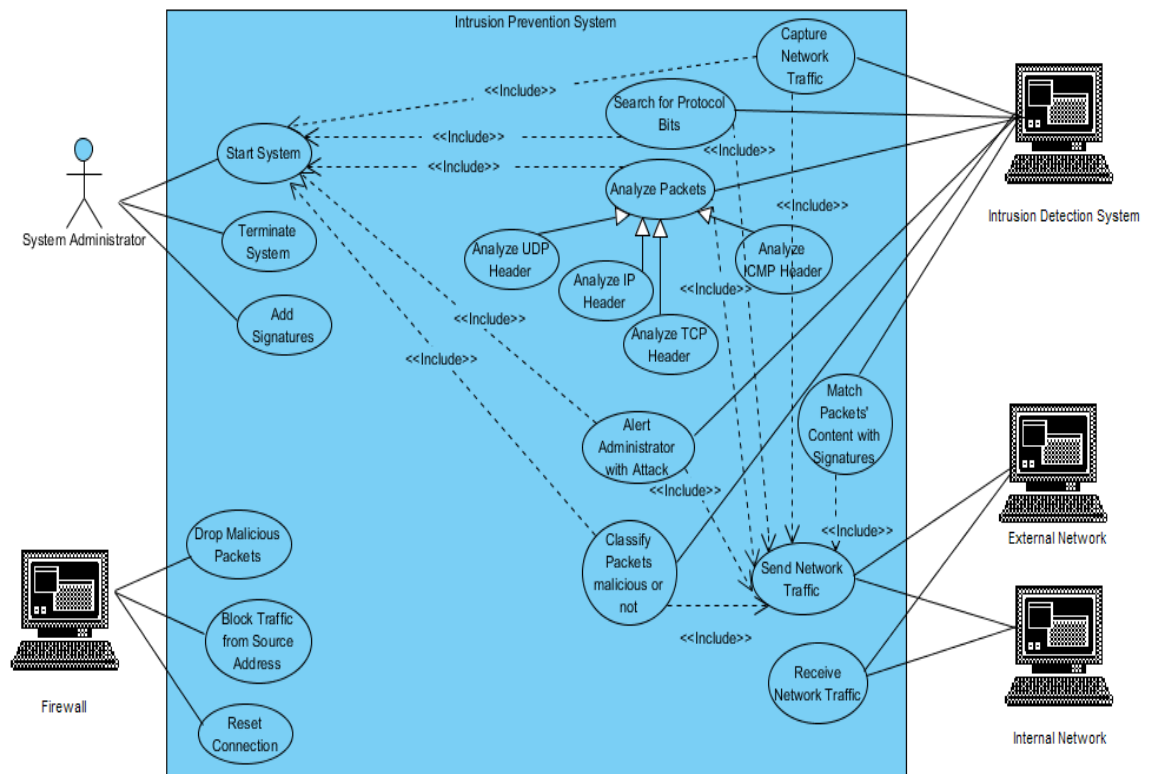


Figure 5 Use case Diagram

System administrator

- Can start and terminate programs.
- Can add signatures to search for in packet payload. But, this requires administrator privilege.

External network

- It deals with the network traffic sent from host.
- It has an intruder detection system implemented inside to check for malicious packets.

Internal network

- It deals with the network traffic sent to host.
- It has an intruder detection system implemented inside to check for malicious packets.

Intrusion Detection System

- It captures the packet.
- It structures the packet and analyzes it.
- It checks for malicious signature or unexpected packet behavior.
- It access firewall to block packets containing malicious data or unexpected behavior.

Firewall

- Access to rules to allow | block | bypass packets with certain values.
- Add rules to firewall.
- Delete rule from firewall.

4.2.3 Proposed Architecture Model

Repository Architecture Model

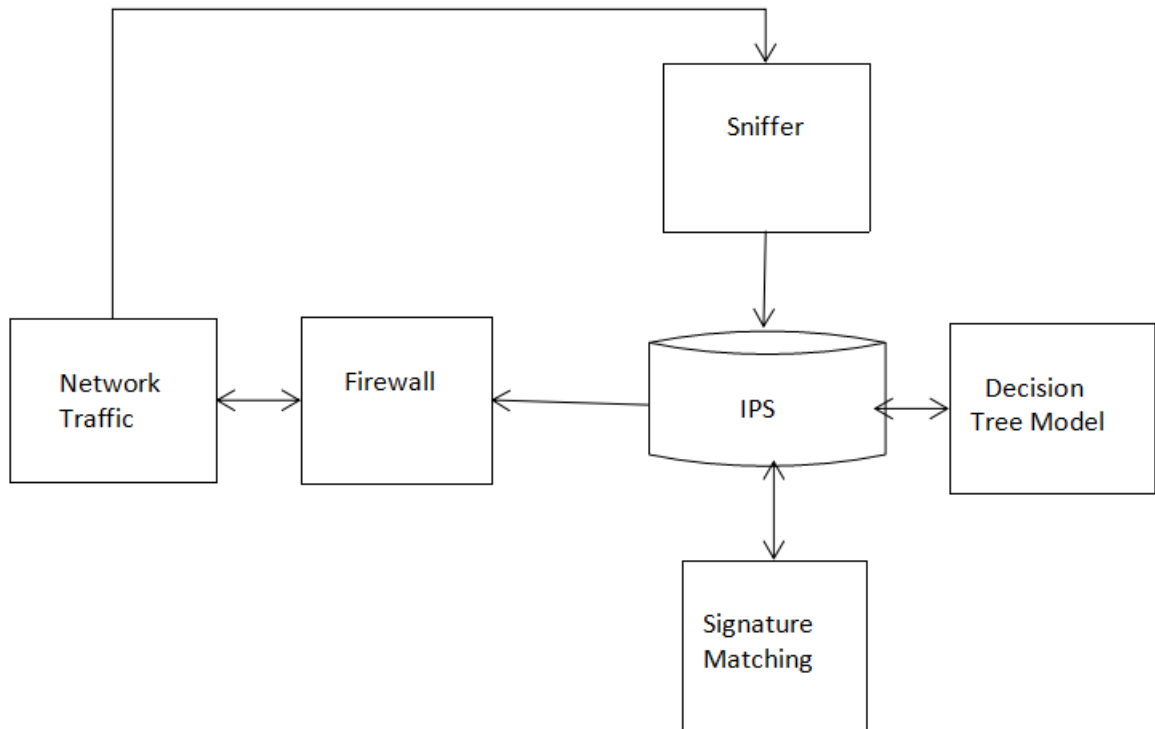


Figure 6 Proposed Repository Architecture Model

- **Network Traffic** passes through firewall and is captured by sniffer system “part of our system” which analyzes traffic with respect to IP structure proposed in RFC and whatever the protocol TCP, UDP or ICMP analyzes it with respect to RFC protocol structures.
- **Sniffer** saves its analyzed data in a structured way in IPS Repository.
- **Decision Tree Model** and **Signature matching** use data and search for signatures in its payload and check if data is normal that obeys standards and there are no anomalies. If there an unexpected behavior either signatures or anomaly one, IPS accesses Firewall add rules, delete rules, reset connection and block network traffic from source address.

4.3 System Design

4.3.1 System Class diagram

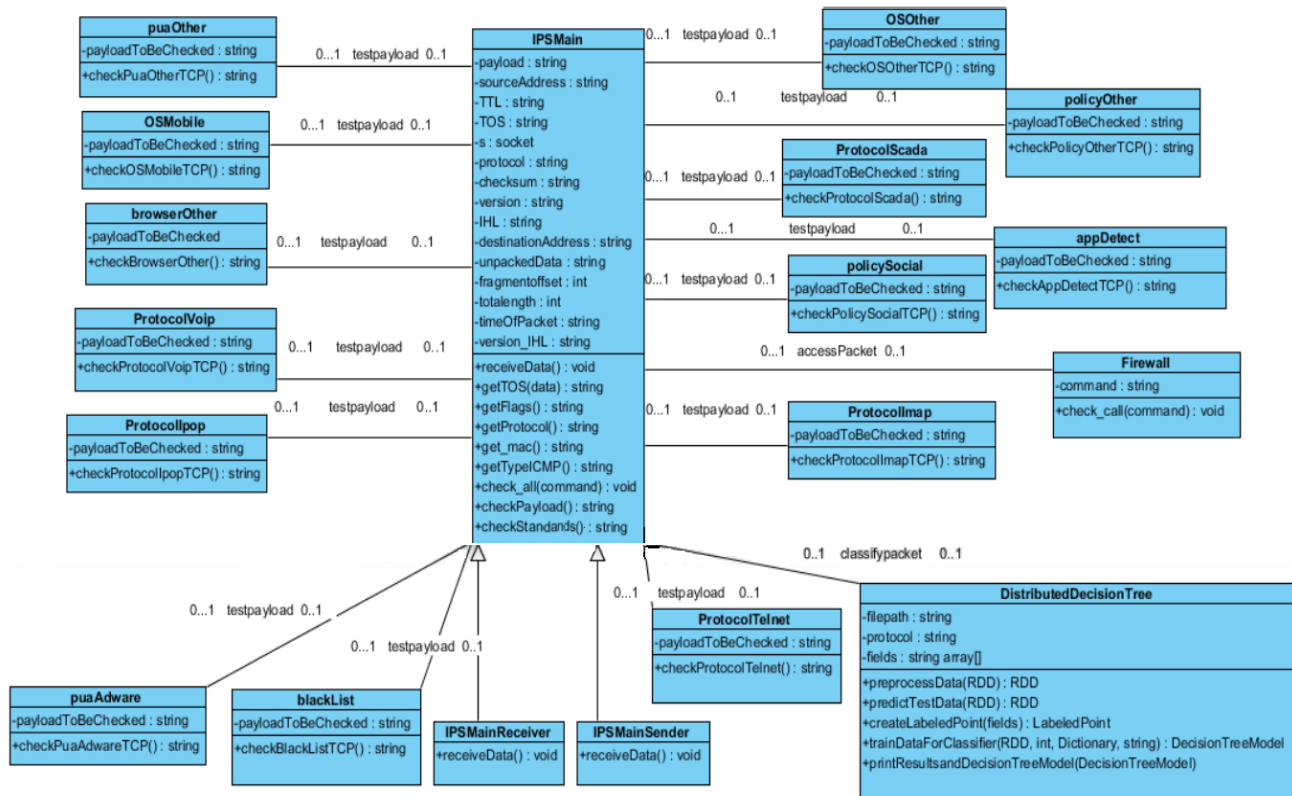


Figure 7 Class Diagram

- We have the **IPSMMain** Class which is the main class in our system where **IPSMMainSender** and **IPSMMainReceiver** inherit from it, this class turns off the firewall, captures network traffic, analyzes it, stores sniffed data in structured way in CSV Files “Datasets” and check that there are no anomalies in packets and the network traffic is normal with respect to RFC standards.

- We have here 14 classes blackList, puaAdware, Protocol Telnet, ProtocolIpop, ProtocolImap, protocolScada, ProtocolVoip, policySocial, puaOther, OSMobile, browserOther, OSOther, policyOther and appdetect which check for signatures in the payload of packets and show alert to user if there are any signatures.
- Firewall class is responsible for adding rules, deleting rules in firewall and resetting connection.
- DistributedDecisionTree class is responsible for classifying network traffic “Packets” as anomalous or not by implementing decision tree algorithm using pyspark API by Spark Big Data Analytics Tool.

4.3.2 System Sequence diagram

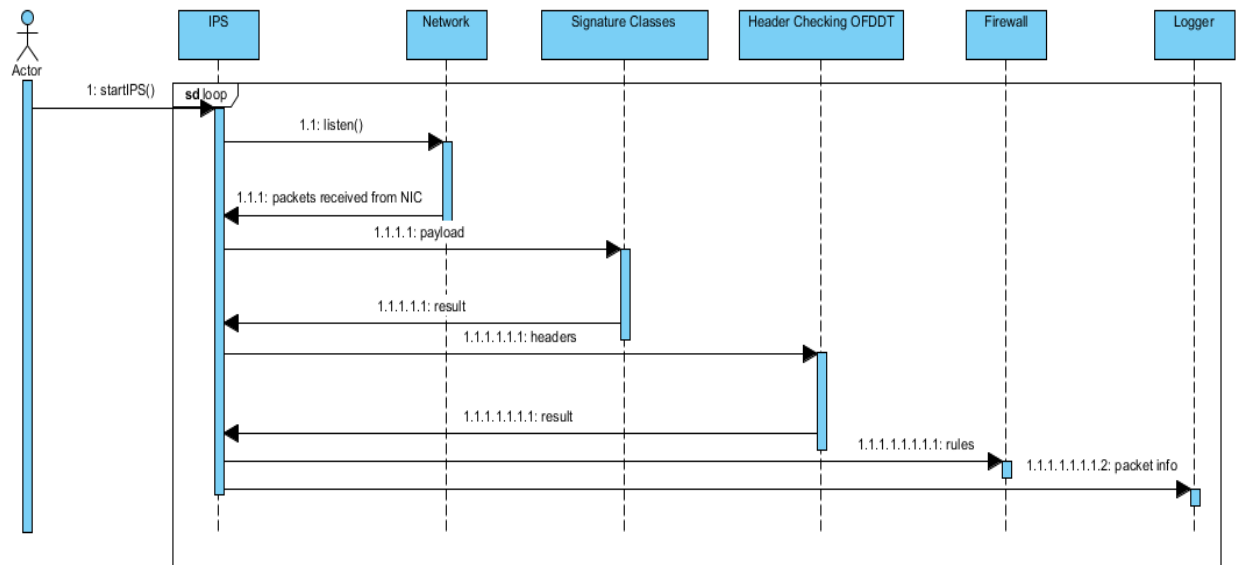


Figure 8 Sequence Diagram

- Upon starting the program, a loop will start in which the program will start listening and wait for any packet to appear on the communication ports. After receiving the packet, the main function will extract headers components and payload the pay load is sent to attack signature classes and the headers components are sent to the standards decision tree. Based on the results the firewall editor will add new rules to the firewall if needed and the logger will record the packet.

5 Project Management

5.1 Used Tools

- 1- We used python language, PyCharm IDE and Wing IDE.
- 2- We started to use Apache Spark framework.
- 3- We used Microsoft Project, Visual Paradigm and Visio tools.

5.2 Teamwork

All team members have worked with each other to achieve effective teamwork during the project .The following decisions are made regarding the teamwork:

- 1- Every sometime there is a meeting held on ground.
- 2- A google drive folder was shared between all team members. to keep project files updated and report up to date.
- 3- Facebook group and a chat group were made for constant communications.

5.3 Planning

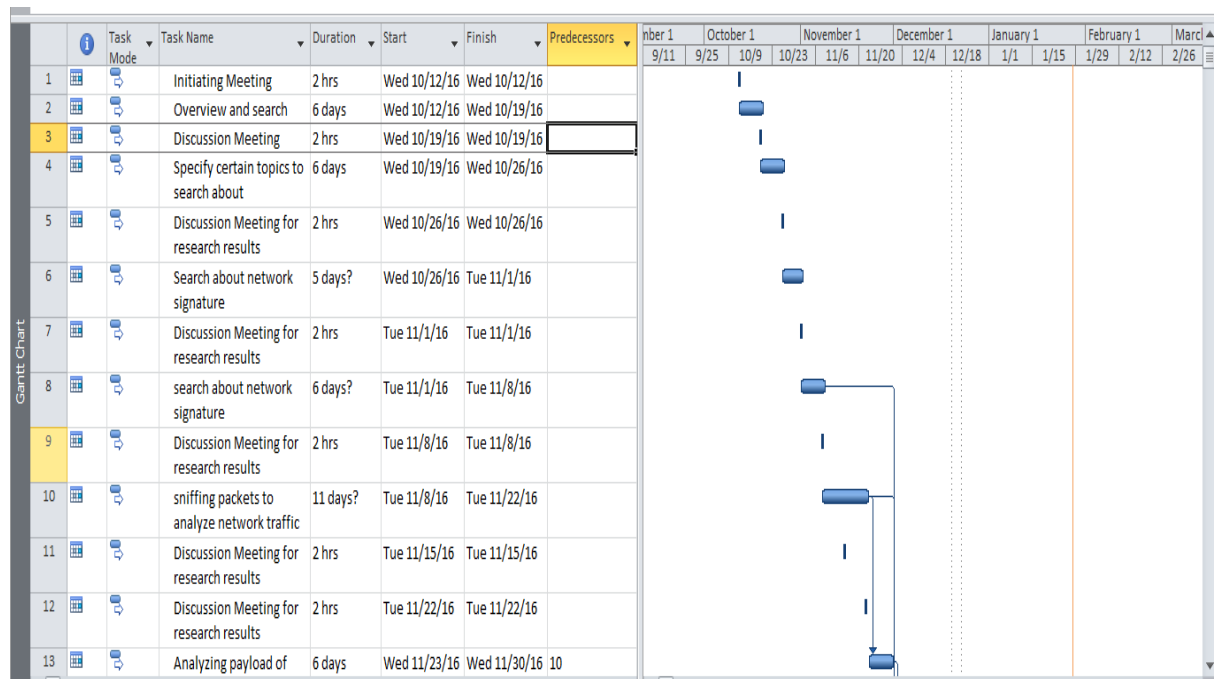


Figure 9 Gantt chart Part 1

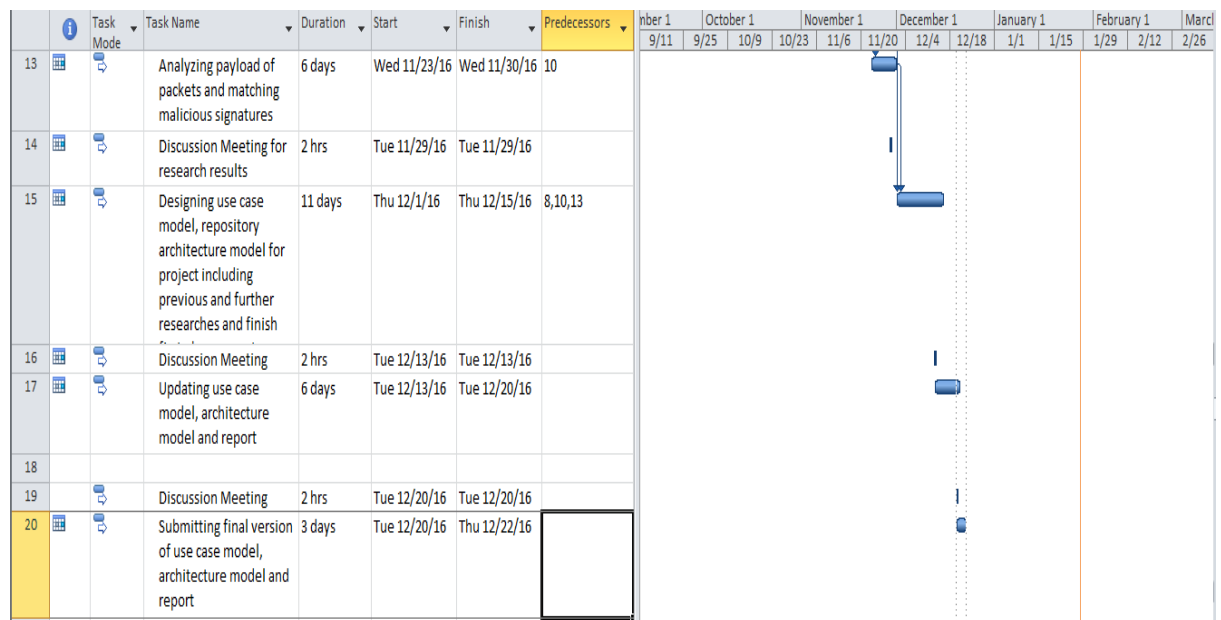


Figure 10 Gantt chart Part 2

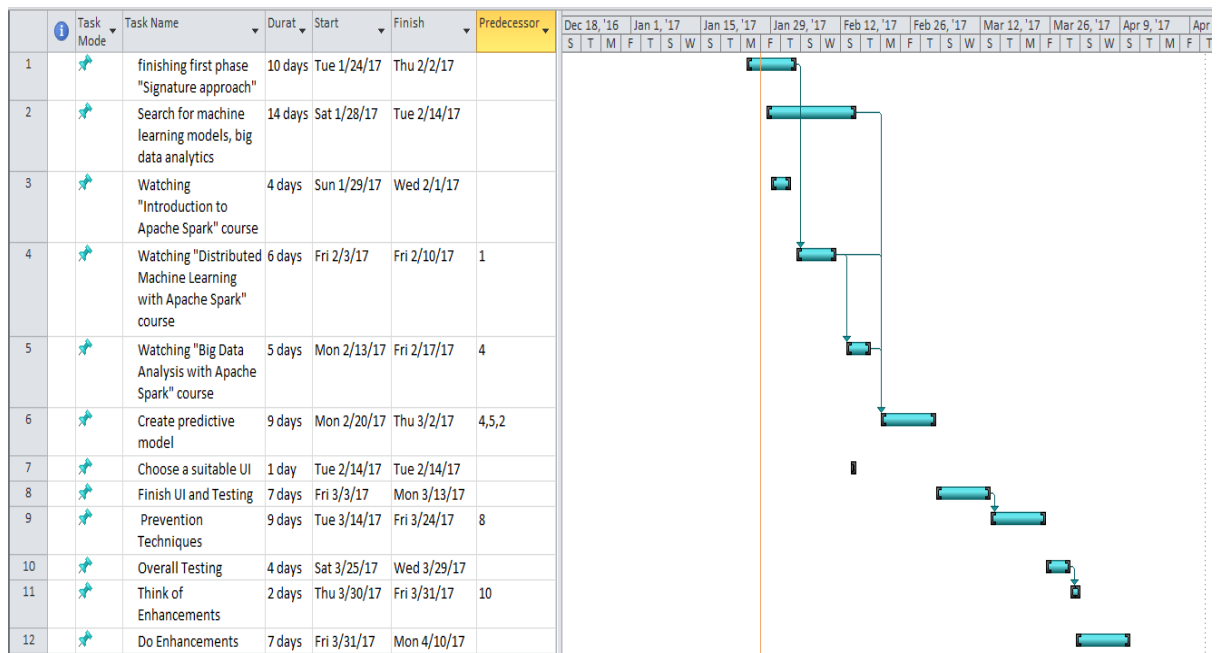


Figure 11 Gantt chart Part 3



Figure 12 Gantt chart Part 4

6 Project Data Analytics Lifecycle

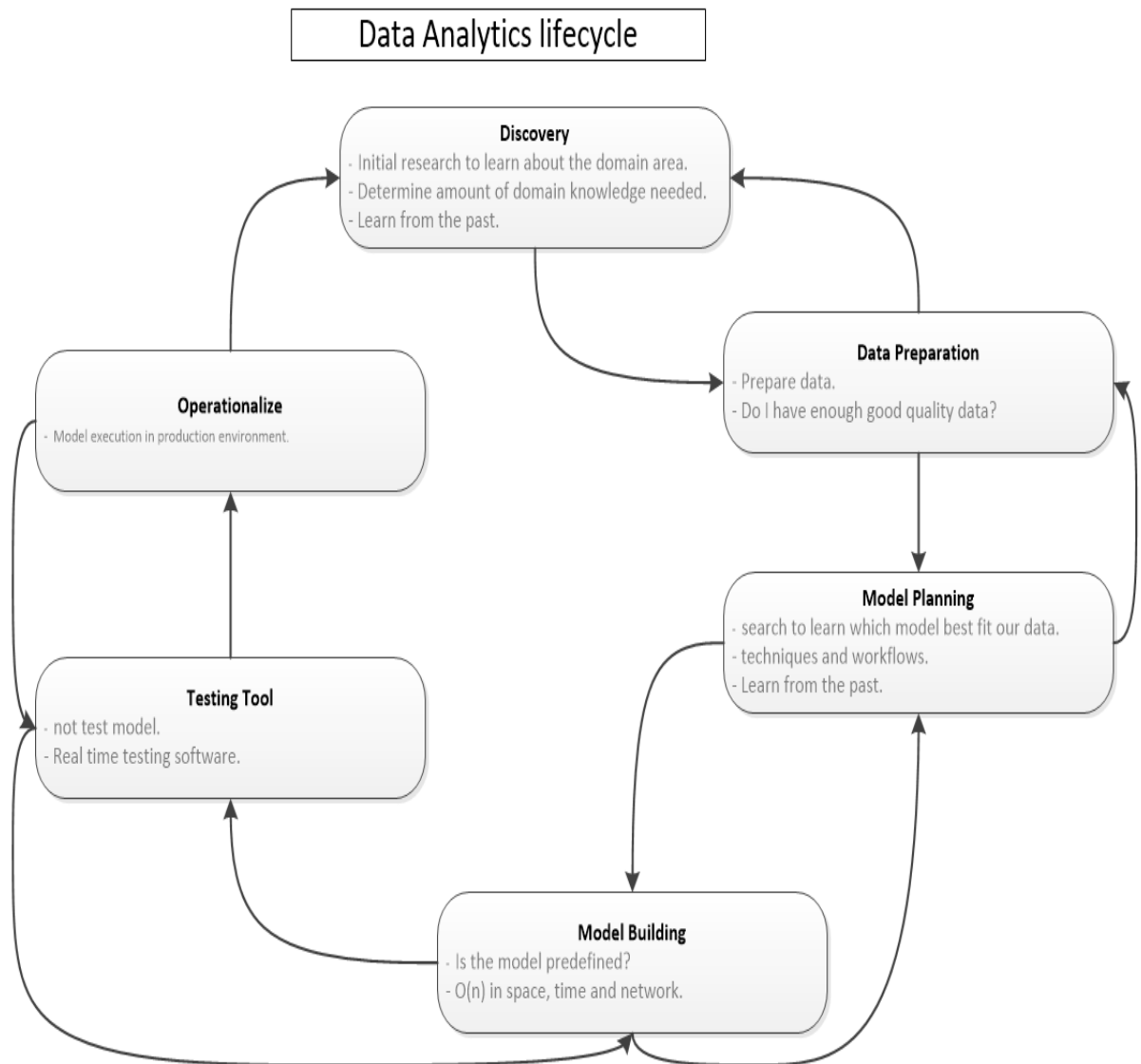


Figure 13 Data Analytics Lifecycle

6.1 Discovery

6.1.1 Search about Intrusion Prevention System

- IPS has two defined approaches: Signature based system and Anomaly based system.
- Socket programming and network structure segmentation are a prerequisite for implementation of IPS.
- IPS is type of security application which secure network by analyzing network traffic and accessing firewall to control network traffic.

6.1.2 Learn from the past

- Search if there have been previous attempts to solve this problem.

In this project we used snort open source signature based IDS to study their role in security field intruder detectors. They are focusing on application layer in network reference model. We used their signature based system.

- Why are we trying again?

Snort IDS is concerned with signatures extracted from packets that contain malicious data. After, requesting a security expert view on system because of unknown system behavior. It's a must that the system is on the internet as IDS or IPS are responsible for analyzing network traffic "data in transmission"

We are doing this project as Snort only used signature-based approach to implement IDS. We are using signature-based approach beside anomaly detection approach. Anomaly detection system is implemented using various machine learning modules to optimize the detection of malicious packets from previous data of malicious and good packets without the need for security experts to specify signature for an attack.

The trend to implement machine learning module was to analyze malicious and good packets and extract feature suitable for packet classification. Naïve Bayesian and decision trees are the most famous modules. These modules are based on features extracted from malicious packets that can come from different cyber-attacks,

malwares, viruses, worms etc... Nowadays attacks are increasing. So, modules are updated and upgraded with the new features needed to detect the malicious packets. Here is the problem that we will try to solve according to our famous phrase “abnormal is not normal”.

We will implement SCA (Security Closed Assumption) module that is supposed to be the newest best module. As, best performance since new entropy detection method is discussed and least features are used to detect the malicious packets. This module is implemented to keep pace with networking evolution as Internet of Things.

6.1.3 Resources

- WINGWARE python IDE, PYCHARM python IDE.
- Cloud community data bricks data analytic platform.
- Group of 4 engineers working parallel.

6.1.4 Frame the problem

- We need security expert and expert in network domain.
- Understand network standards as RFC (Request for Comments) for IETF (Internet Engineering Task Force) and use standard official websites for network standards as IANA (Internet Assigned Numbers Authority)
- Capture the packets.
- Analyzing packet according to case study.

6.1.5 Formulation of Initial Hypothesis

- Implement module for packet sniffing and structuring as the predefined modules are not up to date.
- Structure parts of packets to be analyzed as network and transport layers headers for the machine learning module and the payload of packets to search for malicious signatures from signature based system.

Is the packet good or malicious? H_0 : it's malicious H_A : it's good

Initially we try to prove the packet is malicious, if we fail to prove the null hypothesis we accept the alternative hypothesis that the packet is good, our approach eliminates the chances of false positives.

Table 1 test outcomes

If it's →, and we say it's ↓	Good	Malicious
Malicious	Type I – false positive	OK – true positive
Good	OK – true negative	Type II – false negative

6.2 Data Preparation

6.2.1 Capturing packet

- Create socket and bind it to the local host IP address and port zero to sniff the packets outgoing from host.
- Create socket and bind it to any remote IP address sending packets to local host.

Example

```
b'E\x00\x01\x10o\xc2\x00\x00\xff\x11\x00\x00\xa9\xfe86\xe0\x00\x00\xfb\x14\xe9\x14\xe9\x00\xfcmo\x00\x00\x84\x00\x00\x00\x00\x04\x00\x00\x00\x03\x0254\x0256\x03254\x03169\x07in-addr\x04arpa\x00\x00\x0c\x80\x01\x00\x00\x00x\x00\x0c\x04ziko\x05local\x00\x016\x013\x018\x013\x014\x012\x015\x017\x01B\x01C\x013\x015\x013\x011\x014\x015\x010\x010\x010\x010\x010\x010\x010\x010\x010\x010\x010\x010\x018\x01E\x01F\x03ip6\xc0"\x00\x0c\x80\x01\x00\x00\x00x\x00\x02\xc02\xc02\x00\x01\x80\x01\x00\x00\x00x\x00\x04\xa9\xfe86\xc02\x00\x1c\x80\x01\x00\x00\x00x\x00\x10\xfe\x80\x00\x00\x00\x00\x00T\x13S\xcbu$86\xc0\x0c\x00/\x80\x01\x00\x00\x00x\x00\x06\xc0\x0c\x00\x02\x00\x08\xc0>\x00/\x80\x01\x00\x00\x00x\x00\x06\xc0>\x00\x02\x00\x08\xc02\x00/\x80\x01\x00\x00\x00x\x00\x08\xc02\x00\x04@\x00\x00\x08'
```

6.2.2 Splitting packet fields

- Packets are unstructured data transferred over network.
- IETF made standards and collected those in RFC that structure packet into fields have significant meaning.
- We structured packet according to standards discussed later.

IP header:

00	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
<u>Version</u>				<u>IHL</u>			<u>Differentiated Services</u>									<u>Total length</u>															
<u>Identification</u>															<u>Flags</u>		<u>Fragment offset</u>														
<u>TTL</u>							<u>Protocol</u>									<u>Header checksum</u>															
<u>Source IP address</u>																															
<u>Destination IP address</u>																															
<u>Options</u> and <u>padding</u> :::																															

Figure 14 IP header [3]

Table 2 Version field in IP header

Field name	Field bits	Field description
Version	4 bits	Specifies the format of the IP packet header.

Table 3 Version bits and Description

Version	Description
0	Reserved.
4	IP, Internet Protocol.
5	<u>ST</u> , ST Datagram Mode.
6	SIP, Simple Internet Protocol. SIPP, Simple Internet Protocol Plus. <u>IPv6</u> , Internet Protocol.
7	<u>TP/IX</u> , The Next Internet.
8	PIP, The P Internet Protocol.
9	TUBA.
15	reserved

Table 4 Internet Header Length field in IP header

Field name	Field bits	Field description
Internet Header Length	4 bits	Specifies the length of the IP packet header in 32 bit words. The minimum value for a valid header is 5 and maximum value for a valid header is 15.

Table 5 Differentiated Services field in IP header

Field name	Field bits	Field description
Differentiated Services	8 bits	This field obsoletes the <u>TOS</u> field.

Table 6 Differentiated Services bits

00	01	02	03	04	05	06	07
Code point						Unused	

Table 7 Type of Service Description

Field name	Field bits	Field description
TOS, Type of Service	8 bits	Obsoleted by the Differentiated Services field. This field specifies the parameters for the type of service requested. The parameters may be utilized by networks to define the handling of the datagram during transport.

Table 8 Type of Service bits

00	01	02	03	04	05	06	07
Precedence			D	T	R	M	0
			Delay	Throughput	Reliability	Monetary cost	

Table 9 Precedence field values

Value	Description
0	Routine
1	Priority
2	Immediate
3	Flash
4	Flash override
5	CRITIC/ECP
6	Internetwork control
7	Network control

Table 10 D bit values

Value	Description
0	Normal delay.
1	Low delay.

Table 11 T bit values

Value	Description
0	Normal throughput.
1	High throughput.

Table 12 R bit values

Value	Description
0	Normal reliability.
1	High reliability.

Table 13 M bit values

Value	Description
0	Normal monetary cost.
1	Minimize monetary cost.

Table 14 Total length field in IP header

Field name	Field bits	Field description
Total length	16 bits	Contains the length of the datagram.

Table 15 Identification field in IP header

Field name	Field bits	Field description
Identification	16 bits	Used to identify the fragments of one datagram from those of another. The originating protocol module of an internet datagram sets the identification field to a value that must be unique for that source-destination pair and protocol for the time the datagram will be active in the internet system. The originating protocol module of a complete datagram clears the MF bit to zero and the Fragment Offset field to zero.

Table 16 Flags bits and its description

Flags bits		
00	01	02
R	DF	MF
Reserved	Don't fragment	More fragments
Should be cleared to 0.	Controls the fragmentation of the datagram.	Indicates if the datagram contains additional fragments.

Table 17 Don't fragment bit values

Value	Description
0	Fragment if necessary.
1	Do not fragment.

Table 18 More fragments bit values

Value	Description
0	This is the last fragment.
1	More fragments follow this fragment.

Table 19 Fragment Offset field in IP header

Field name	Field bits	Field description
Fragment Offset	13 bits	Used to direct the reassembly of a fragmented datagram.

Table 20 Time to live field in IP header

Field name	Field bits	Field description
Time to Live	8 bits	A timer field used to track the lifetime of the datagram. When the TTL field is decremented down to zero, the datagram is discarded.

Table 21 Protocol field in IP header

Field name	Field bits	Field description
Protocol	8 bits	This field specifies the next encapsulated protocol.

Table 22 Protocol field Values analyzed in project

Value	Protocol	References
1	<u>ICMP</u> , Internet Control Message Protocol.	<u>RFC 792</u>
6	<u>TCP</u> , Transmission Control Protocol.	RFC 793
17	<u>UDP</u> , User Datagram Protocol.	RFC 768

Table 23 Header checksum field in IP header

Field name	Field bits	Field description
Header checksum	16 bits	A 16 bit one's complement checksum of the IP header and IP options.

Table 24 Source IP address field in IP header

Field name	Field bits	Field description
Source IP address	32 bits	IP address of the sender.

Table 25 Destination IP address field in IP header

Field name	Field bits	Field description
Destination IP address	32 bits	IP address of the intended receiver.

If the protocol field in IP header states that this protocol is TCP we partition payload of IP header as

TCP header:



Figure 15 TCP header [4]

Table 26 Source Port field in TCP header

Field name	Field bits	Field description
Source Port	16 bits	Identifies the sending port.

Table 27 Destination Port field in TCP header

Field name	Field bits	Field description
Destination Port	16 bits	Identifies the receiving port.

Table 28 Sequence Number field in TCP header

Field name	Field bits	Field description
Sequence Number	32 bits	The sequence number of the first data byte in this segment. If the SYN bit is set, the sequence number is the initial sequence number and the first data byte is initial sequence number + 1.

Table 29 Acknowledgment Number field in TCP header

Field name	Field bits	Field description
Acknowledgment Number	32 bits	If the ACK bit is set, this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.

Table 30 Data Offset field in TCP header

Field name	Field bits	Field description
Data Offset	4 bits	The number of 32-bit words in the TCP header. This indicates where the data begins. The length of the TCP header is always a multiple of 32 bits

Table 31 Reserved field in TCP header

Field name	Field bits	Field description
Reserved	3 bits	Must be cleared to zero.

Table 32 ECN, Explicit Congestion Notification bits

00	01	02
<u>N</u>	<u>C</u>	<u>E</u>
N, NS, Nonce Sum	C, CWR	E, ECE, ECN-Echo
This is an optional field added to ECN intended to protect against accidental or malicious concealment of marked packets from the TCP sender.		

Table 33 Control bits

Control Bits					
00	01	02	03	04	05
<u>U</u>	<u>A</u>	<u>P</u>	<u>R</u>	<u>S</u>	<u>F</u>
U, URG	A, ACK	P, PSH	R, RST	S, SYN	F, FIN
Urgent pointer valid flag.	Acknowledgment number valid flag.	Push flag.	Reset connection flag.	Synchronize sequence numbers flag.	End of data flag.

Table 34 Window field in TCP header

Field name	Field bits	Field description
Window	16 bits-unsigned	The number of data bytes beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept.

Table 35 Checksum field in TCP header

Field name	Field bits	Field description
Checksum	16 bits	This is computed as the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the TCP header, and the data, padded as needed with zero bytes at the end to make a multiple of two bytes

Table 36 Urgent Pointer field in TCP header

Field name	Field bits	Field description
Urgent Pointer	16 bits-unsigned	If the <i>URG</i> bit is set, this field points to the sequence number of the last byte in a sequence of urgent data.

If the protocol field in IP header states that this protocol is UDP we partition payload of IP header as

UDP header:



Figure 16 UDP header [5]

Table 37 Source Port field in UDP header

Field name	Field bits	Field description
Source Port	16 bits	The port number of the sender. Cleared to zero if not used.

Table 38 Destination Port field in UDP header

Field name	Field bits	Field description
Destination Port	16 bits	The port this packet is addressed to.

Table 39 Length field in UDP header

Field name	Field bits	Field description
Length	16 bits	The length in bytes of the UDP header and the encapsulated data. The minimum value for this field is 8.

Table 40 Checksum field in UDP header

Field name	Field bits	Field description
Checksum	16 bits	Computed as the 16-bit one's complement of the one's complement sum of a pseudo header of information from the IP header, the UDP header, and the data, padded as needed with zero bytes at the end to make a multiple of two bytes. If the checksum is cleared to zero, then check summing is disabled.

If the protocol field in IP header states that this protocol is ICMP we partition payload of IP header as

ICMP header:



Figure 17 ICMP header [6]

Table 41 Type field in ICMP header

Field name	Field bits	Field description
Type	8 bits	Specifies the format of the ICMP message.

Table 42 Type field values [11]

Type	Name	Reference
0	Echo Reply	[RFC792]
1	Unassigned	
2	Unassigned	
3	Destination Unreachable	[RFC792]
4	Source Quench (Deprecated)	[RFC792][RFC6633]
5	Redirect	[RFC792]
6	Alternate Host Address (Deprecated)	[RFC6918]
7	Unassigned	
8	Echo	[RFC792]
9	Router Advertisement	[RFC1256]
10	Router Solicitation	[RFC1256]
11	Time Exceeded	[RFC792]
12	Parameter Problem	[RFC792]
13	Timestamp	[RFC792]
14	Timestamp Reply	[RFC792]
15	Information Request (Deprecated)	[RFC792][RFC6918]
16	Information Reply (Deprecated)	[RFC792][RFC6918]
17	Address Mask Request (Deprecated)	[RFC950][RFC6918]
18	Address Mask Reply (Deprecated)	[RFC950][RFC6918]
19	Reserved (for Security)	[Solo]

20-29	Reserved (for Robustness Experiment)	[ZSu]
30	Traceroute (Deprecated)	[RFC1393][RFC6918]
31	Datagram Conversion Error (Deprecated)	[RFC1475][RFC6918]
32	Mobile Host Redirect (Deprecated)	[David_Johnson][RFC6918]
33	IPv6 Where-Are-You (Deprecated)	[Simpson][RFC6918]
34	IPv6 I-Am-Here (Deprecated)	[Simpson][RFC6918]
35	Mobile Registration Request (Deprecated)	[Simpson][RFC6918]
36	Mobile Registration Reply (Deprecated)	[Simpson][RFC6918]
37	Domain Name Request (Deprecated)	[RFC1788][RFC6918]
38	Domain Name Reply (Deprecated)	[RFC1788][RFC6918]
39	SKIP (Deprecated)	[Markson][RFC6918]
40	Photuris	[RFC2521]
41	ICMP messages utilized by experimental mobility protocols such as Seamoby	[RFC4065]
42-252	Unassigned	
253	RFC3692-style Experiment 1	[RFC4727]

254	RFC3692-style Experiment 2	[RFC4727]
255	Reserved	[JBP]

Table 43 Code field in ICMP header

Field name	Field bits	Field description
Code	8 bits	Further qualifies the ICMP message.

Table 44 Type 0 in code field

Type 0		
Codes	Description	Reference
0	No Code	

Table 45 Type 3 in code field

Type 3		
Codes	Description	Reference
0	Net Unreachable	[RFC792]
1	Host Unreachable	[RFC792]
2	Protocol Unreachable	[RFC792]
3	Port Unreachable	[RFC792]
4	Fragmentation Needed and Don't Fragment was Set	[RFC792]
5	Source Route Failed	[RFC792]

6	Destination Network Unknown	[RFC1122]
7	Destination Host Unknown	[RFC1122]
8	Source Host Isolated	[RFC1122]
9	Communication with Destination Network is Administratively Prohibited	[RFC1122]
10	Communication with Destination Host is Administratively Prohibited	[RFC1122]
11	Destination Network Unreachable for Type of Service	[RFC1122]
12	Destination Host Unreachable for Type of Service	[RFC1122]
13	Communication Administratively Prohibited	[RFC1812]
14	Host Precedence Violation	[RFC1812]
15	Precedence cutoff in effect	[RFC1812]

Table 46 Type 5 in code field

Type 5		
Codes	Description	Reference
0	Redirect Datagram for the Network (or subnet)	
1	Redirect Datagram for the Host	
2	Redirect Datagram for the Type of Service and Network	
3	Redirect Datagram for the Type of Service and Host	

Table 47 Type 8 in code field

Type 8		
Codes	Description	Reference
0	No Code	

Table 48 Type 9 in code field

Type 9		
Codes	Description	Reference
0	Normal router advertisement	[RFC3344]
16	Does not route common traffic	[RFC3344]

Table 49 Type 10 in code field

Type 10		
Codes	Description	Reference
0	No Code	

Table 50 Type 11 in code field

Type 11		
Codes	Description	Reference
0	Time to Live exceeded in Transit	
1	Fragment Reassembly Time Exceeded	

Table 51 Type 12 in code field

Type 12		
Codes	Description	Reference
0	Pointer indicates the error	
1	Missing a Required Option	[RFC1108]
2	Bad Length	

Table 52 Type 13 in code field

Type 13		
Codes	Description	Reference
0	No Code	

Table 53 Type 14 in code field

Type 14		
Codes	Description	Reference
0	No Code	

Table 54 Type 40 in code field

Type 40		
Codes	Description	Reference
0	Bad SPI	
1	Authentication Failed	
2	Decompression Failed	
3	Decryption Failed	
4	Need Authentication	
5	Need Authorization	

Table 55 ICMP Header Checksum field

Field name	Field bits	Field description
ICMP Header Checksum	16 bits	Checksum that covers the ICMP message. This is the 16-bit one's complement of the one's complement sum of the ICMP message starting with the Type field. The checksum field should be cleared to zero before generating the checksum.

6.2.3 Creating datasets

Example of TCP train dataset

Table 56 TCP Dataset Part 1

Time Of Packet	version	Internet Header Length	Type Of Service	Total Length
38:36.0	4	20	Routine Normal delay Normal throughput Normal Reliability Normal monetary cost	52
41:59.0	4	20	Priority Normal delay high throughput Normal Reliability Normal monetary cost	536
43:19.4	4	20	Priority Normal delay high throughput Normal Reliability Normal monetary cost	52
59:15.2	4	20	Routine Normal delay Normal throughput Normal Reliability Normal monetary cost	1452

Table 57 TCP Dataset Part 2

Identification	Flags	Fragment Offset	Time To Live	Protocol	Check Sum
0xea72	0 - Reserved bit 1 - Do not fragment 0 - last fragment	0	57	Transmission Control Protocol	32995
0xb8e0	0 - Reserved bit 1 - Do not fragment 0 - last fragment	0	53	Transmission Control Protocol	9588
0x55f1	0 - Reserved bit 1 - Do not fragment 0 - last fragment	0	53	Transmission Control Protocol	35399
0x7e8d	0 - Reserved bit 1 - Do not fragment 0 - last fragment	0	58	Transmission Control Protocol	1786

Table 58 TCP Dataset Part 3

Src. Add Oct1	Src. Add Oct2	Src. Add Oct3	Src. Add Oct4	Dst. Add Oct1	Dst. Add Oct2	Dst. Add Oct3	Dst. Add Oct4	Options And Padding
23	51	253	144	192	168	1	2	
151	101	12	84	192	168	1	2	
151	101	12	84	192	168	1	2	
69	16	175	10	192	168	1	2	

Table 59 TCP Dataset Part 4

Source Port	Destination Port	Sequence Number	Acknowledgment Number	Data Offset	Reserved TCP	Notification	Congestion
443	54331	5.63E+08	6816382	8	0	0	0
443	54441	1.23E+09	3.71E+08	8	0	0	0
443	54512	1.43E+09	1.63E+09	8	0	0	0
80	59550	1.27E+09	2.52E+09	5	0	0	0

Table 60 TCP Dataset Part 5

Explicit	Control U	Control A	Control P	Control R	Control S	Control F	Window	Check Sum TCP	Urgent Pointer
0	0	1	0	0	0	0	946	26758	0
0	0	1	1	0	0	0	95	61911	0
0	0	1	0	0	0	0	76	11367	0
0	0	1	0	0	0	0	31	31947	0

Table 61 TCP Dataset Part 6

Options TCP	Payload TCP	Signature TCP	Reason Of Signature Status
b'\x01\x01\x05\n\x00h\x02^\x00h\x02~'	b''	Good	OK
b'\x01\x01\x05\n\x16\x18\x9c\x8d\x16\x18\x9c\x8e'	b'\x06.\xb9\xaf\x184- 9\x04\xb5\xdc2c"\x8f>\x19\xba\xdd\x7feF\x922\xfd/\x00:\x03\x08\x96\xfa\xca\xbd\x857\xd9\xd7\xe7\xc4\xdb\x04\xe8hh\x9f\x07\x93\xb6X&F\x13)\x18\xe8\xa5\xfaV' \x1d\xfa=\xb5*n\xa8\xbf\xde5!g\x059\xb9\xd1C\x84+\xd0\xec\xfae\x93\x92@aFCOD#\x8bj^\xfd8\xd5*U)\xae\xb7\x8f\x02\x8e#\x80\xab~;SA8O\xbe\x15a\xca\x8d\xe6\x01\xa5>\xa7gF\x1aU1\xdaA\\z\rq\xcd\x99\\\xbf\x9b 05dV\xd0\x8a\xef:\xe5\x87\x8a\x03\x05\x1f\xbb1=\x8f\xc2P\xec\x11\xa8\x87\xefcl\xfa9\x9e!\x94\xfd\xdb\xcb\xa1\xd197\x02\x82\x85\xe6\xd4\x84\xb7C\t\xb9l\xfa7\xca\tGG\x87\xb3 =G\xb4O\xc4^\xc5\xb2\x9d\x1a\xfa5\x88Re\x11T\xed\xfa4\xfd\xfa3!\xcc)\xb5#:N\xcc1\x142H\x16\xe6\xb3x\xe4@\xfe\xbe\x0f5\xbe\x17\xca4\xb3g\x82\x95_tk\x0b\xfa\x0f\x8b\xcf\x9a5\xa2\xd6\xad\xfdXXo\x8f\xb0d\x12\x88P\x80\x18\xe2CHw\xff:\xf6O\n\x99u\xe5\xcc\x88 \xb8\xaf\xdc\xfae\x14\x0fC\xe8\xaa\xe1\x1bz\x14\x13\xcf\x958\xe0\x9ezE\xb3\xe6\x12P=\x16\xfa7\xd2?w\xb7\xfa5G\xa1\xa2\xe5)TA8\x08\xec\xca6\r\xfa1\	Good	OK

	xad\xa2 \xcb\xc3\xa4,Z\xb7ea)\xbe\xc6\xa1\x00\x99\r\x86(\xb4Z_e\x1a\x8c\x03\x97\x16G?_ \xa7_\xf7\xdf\xbb1\x86\xb3\xda\xa0\x9d\xe1&\x95\xb7\xc5L#\xe4\ca\x8b_\xa6\xb7\xaeX\xae\x04\xfcq\x97\x97\xc6\x5f\x92\xffU_\x8f\xf7\xd0T\xb5\x03\r\r\xca+\$_O\r\x7f=B!\x8c\xf7X\x1a\xc4\xee!=N#\xa6Prv\xbe`\$+\xc0v0ms\x2\x21\x0c^\xf6\xc1\xc7\xf1\x02d\x97\xa6\x94\x8d\x88*\xf0"{{\x8f\xd3N\xc6l\x10\x9d{2\xf3Z+\r\x0f\xc4\xbb\xc8'		
b'\x01\x01\x05\naQn\x91aQn\x92'	b''	Good	OK
	b'\xdf\x0e~U\xb55\xe7\xad\xe73\x91^^\x93\x8a6\xc9\xdb\xe7k\xe4DW:y\xe4Fs\xc3\xb5JN\xf1\xb3T\x03\xb4\xdd\x8e\x1af\x9c\xf7\x16\xecm\x873j\xbc\xd7\x8f\x7fB\nl\'^\x85Y\x16\x08\x89b\xaf\xec\xf1j0C\x94\x0c\xda,Z\xda3\x1ctc\xb9W3\x15\x00\xefq\x86\x04\x1c\xec\xe6\xd5\xa1\x9d\xfe\x9eZ\x85\xbf\xd7\xc8\xcf\xd8\xcc\xdf\xbf\xb1\xe5\x97\xa7Ti^\x93f\xc4\x92\xc6\xcf\xe7\xfb}\xf3\x97\xbf\xfc\xfd\xf3\xef\xf6\xf5\xfb\xe7\xbe}\xbd\xf9\xfaE\xb3\xaay\x8a\xfd5\x19b&\xa5\xfc\xc37\xbc\x93\xdf\x7fx\$\xfd9\xfd\x98\xff\x00cl\x1c\xb4\xfaJ\xff\x00\xe6zL\xee7_VtV\x99\xf9\xa7+^Y\xfbax8dAV\xd61\xf1\xf7D\x8eXD`\xf7\xa0U\xcdT\xc16;\xb2\xd5\x87\xb5\xbb8\x85\xb6,\xc6\xc9\x94\x1a\xefG\xaf\x14\xa9\xd3d\xa2\x08\xa71\x82\xad\xb5QR<\xa7h\xa0\xa8,\xa8\x84\x1b\xfb6\xc8\ a\xd0\xab[\xf9\x0eck@\x1a\xe2\x08\xe7\xa9[\xfd \x9c~oP/N\x90g(\xc2\xf7)e9\xd8?B\xdc\xab\x11=	BAD	('Alert!!!\t', 'POLICY-SOCIAL multiple chat protocols link to local file attempt')

<p>Y\xcb\xdb\xef\x9f\x7f\xb4 \x9c\xbd9z}\xb3\xed\x7f\xfb bs\xce\xac\xa4\xf2\xc8')\x8b\xfb\xec\xed\xfa\xbfN\xfb f\x00\xd8\x7f\xf3\xea)\xcb\x86\xf4\xff\x00\xc6d\xdf:s \x7f\xc8\xa4G\xa4\xcfLwnL\xa8(<z\xc5(X\xd4:k\x07Pn \x9d\xac\x05L\x8c\xeb\xe5\x03\xa7Tj\xd7\xb0e\xb7\xbcW \xe91j)\xfa\x18U(\x925\xf5MC%bB\xb8Dk\x03\xab #\xb8H\x9b\x05H#&k\x15K\xdc\xe6]\x18t\xc76\xc7\x89? \x88S\xfd\x7f^^\x9c\xbd\xf2\x7f\x7f\xa0`A\xb3\x94Fr \x9yz\xf2\xf4\xe5\x9f \xfb\xfd\xbe^~\x9c\xfd\x9e \xb73}2\xc71\x9b\xfc\xcb\x0b\xb3\xc2zy\xe5\xab\xf7\x xb6s\x88\xc8\xaf\xad\x89\xef\x02\xe7=Y\x14\x88\xce \xe5\xaf4\x0cVx\x83a\xe1\xa7\xfa\x8c\x9d\xe1T\x1e\x9a \xa0L\x9e\x80\xb9s\x88\rX"\x13YO\xd7k\xaf\x1e\x c1dnB\xe5\n*D\x0cb\xc3;bH\x95\xeb\xbb\x1d\xb5\x17] \x87X\xb2\xc3\xe7w\xc3Ql\xf8\x8b\xdbP\xa4\x7fM\x xeb\xf6\xfa\xff\x00\x7fO\xb7\xff\x00\x89\xef\xf6>\x9d \xbfk\xd3\xad\xcd\x98\xfb\x1a\x1d7(\xd6r\xe7\x9c\xbd& \xdc\xb3\x94\xdb=\xa3&\xbc\xf2l\x90)\xb4\xfd\x9b\x1d \xc87*\x15\xc5\rv\xf2\x9d\x07\$G;i\xd4\xa5\x13\xd8\x1e \x15\xa2\xea\xf5\xce\xbf\xa0#\x8fKZ+\x15\xb4Z6\xba\xda ?F\xd2\xf8u\xe8\xd5\xd8\x94\xd4\xb5)w9\xc9GS\xdf\x c5\x18\xea\xc3\x96\x9c8n-\x97\x11O\xfeB_\xaea\x07q\xcf, \xfb\xfd.Y?//\xa7\xcb\xd2~^^\xaf\x7f\xb5\xf5\x17\xd4\xd2 \xbcC\xc4?\xc4\xe9\xbf\x8c\xf4\xf7\x9c\x88\x8c\x98\xcbZ+ \x9c\xacL\x8a\xc4zL\xe3\x8c\x0c0\xd7u\xdb\xbb{Z\xe3\x17 \xf4\xba\xc5\xc3.\x87`\x12\xb7j\xd2\xf9eC\xca\xb1Jd\xccV /\xb8\xad\xc9\xe3\x15\xc6\xeb\x11X)j8hR\xc5\xed\xafj \xe6\xf2\x19#U\\\x85\xa5\x98\x18D1\x14\xf3b\xae\x85I '\xbe\xdb\x89?\x89O\xfdLh\xbd\xfbz\xf2\xf4\xbd\xeb[\xdc\xbeNY\xf7\xe5\xf2\xfb\xce^\x91\xe9\xcb\x</p>		
---	--	--

<p>xe7\x8e\\\xfd\x1e\xff\x00{\xeaEb-</p> <p>\xbf\xfe\'K\xfc^r\x4\x99\xe5\x9c\xed\xadb\xbf!#\xb9</p> <p>[\xea\xd7\xbe3\xa5\x11"4\xc0\xaeq\x01b\xcdV\xcc7}: uIF\xdc\x12\xb4\x0e\xfb\xbe\xddGA\xe5\xba\x0c1\x0 e;\x83\x8e\x9a\xf5NDz\xed\x9b\xa6\xfc\xd8\x8c\xcf5 A\x9d\x6]\xcf\xe95\xd7jZ&\xe3\x89\x7f\x88N?\xa4\x fb\xe7,\xe5\xe9\xcb>\xfd7\xb7\xa7,\xe5\x9c\xb3\x97\ xa7,\xe5\x9c\xbd\xb2\xbf\xbd\xfd\xef\xcb9g,\xe5\x9c \xb3\x96r\xceY\xcb#\xf6\xe5\xef\xcb9g/^Y\xcb9zr\xcd \x87??\xea\xf1\x1f\xf1\x1ao\xe2\xfd&y\xe4W\xe5\xe 5\xeb3\x9b}\x80\x90\x19-</p> <p>b]\x06.\xb3\x07\xdd\x04Ta\x86\xb6\xa7\x00\xe9\xae Y\xad\xff\x00L%\r\xbc\xd2\x00\xb0}-</p> <p>\xdf\xea]\x83l=6A\xa1\xd1Q\x96\x1c\x1a\xe8\xae\x98 \xfa\xdaw%\x85\x92\xb1\xe8\xddw<M\xfcB\x7f\xea\x fc\xff\x00\x7f\x9f\xed\x2}\xbe_\xbe}\xfeO\xbf\x9\x2f 7\x4\xd9\xff\x00\xb9\x1fW\x88\xbf\x88\xd4l\xd3\xa 6\xbf\xe2lg\xc4\x13\xcf\x89%\x9f\x11O>\$\x9e l<\xf8\ x92y\xf1\$\xf3\xe2l\xe7\xc4\x92\xc9\xd8\'8\xc6\xc5J\x 89\xb2\xf9/\x83^\xb4\xc3\xc6\x1do\xd1\xce\x83dz\xc 5\xed\x12i\xa8\x05kp\xfb\x8a\x02\xbez\x99\xe7)\x9e r\xb9\xe7)\x9er\xb9\xe7-\x9b-</p> <p>\xc4_(\xd5\xd5`VM\x00\xc9\xed\xb0\x8f/J\xad\t\xd8\ xb3\x1b\xbe#\xac\xdfR\x9b\x00\x85\xbb\xe0\xce\x2f g;\xc2\xce\xe8\xb3\xba,\xee\x0f;\x83\xc9(\xf9\xf7\x0 7\x9d\xd1\xe7py\xdc\x1ew\x07\x9d\x1\xe7py\xdc\xa 6u\x8f;\x83\xce\xb1\xe7]9\xf5\x8f:\xe9\x9dt\xce\xbag X\xf3\xae\x99\xd5\\\xeb\xa6u\xd3:\xa9\x9dt\xce\xaa gUs\xaa\x99\xd5\\\xea\xa6uW:\xab\xcb\xaa\x99\xd5\ \\\xea\xa6l\xa6\xbe}RR\xa4\xaf\xc2\x11\xcf\x84#\x9f\ x0bK>\x14\x96F\xb9H\xc9\xd7\xa99\xe0+\x96\xd7)l\x</p>		
--	--	--

	f8ZY\xf0\xb4\xb3\xe1ig\xc3\x13\cf\x83!\xd7\xf0\xc4\xf3\xe1id\xeaR\xe7\xf0\xb4\xba\xa3X\x9cg\xc2\xd3\xe7\xf0\x84y\xfc%\x1c\xf8B9\xf0\x84rt\xe8\xce \x19\x0c\x8d&\xbe'\xe0\xa8s\x9d*\x13\x9f\x02\xd7\xe7\x00\xb5\xd9\xf0\x1d~ \x03_\x80\xd2\xa4\x03^\x90J'		
--	--	--	--

Example of UDP dataset

Table 62 UDP Dataset Part 1

Time Of Packet	version	Internet Header Length	Type Of Service	Total Length
46:48.4	4	20	Routine Normal delay Normal throughput Normal Reliability Normal monetary cost	52
54:31.9	4	20	Routine Normal delay Normal throughput Normal Reliability Normal monetary cost	68
54:57.1	4	20	Routine Normal delay Normal throughput Normal Reliability Normal monetary cost	99

Table 63 UDP Dataset Part 2

Identification	Flags	Fragment Offset	Time To Live	Protocol	Check Sum
0x28ef	0 - Reserved bit 0 - Fragment if necessary 0 - last fragment'	0	128	User Datagram Protocol	0
0xb85	0 - Reserved bit 0 - Fragment if necessary 0 - last fragment'	0	128	User Datagram Protocol	0
0x0	0 - Reserved bit 1 - Do not fragment 0 - last fragment'	0	64	User Datagram Protocol	46811

Table 64 UDP Dataset Part 3

Src. Add Oct1	Src. Add Oct2	Src. Add Oct3	Src. Add Oct4	Dst. Add Oct1	Dst. Add Oct2	Dst. Add Oct3	Dst. Add Oct4
127	0	0	1	127	0	0	1
192	168	225	1	192	168	225	255
192	168	1	92	192	168	1	2

Table 65 UDP Dataset Part 4

Source Port	Destination Port	Length	Check Sum UDP
48600	48601	32	60516
63648	1947	48	17130
53	63072	79	36542

Table 66 UDP Dataset Part 5

payload	Signature	Reason Of Signature Status
b'GFEN\x02\x00\x00\x00\x0b\x00\x00\x00\x00\x00\x00\x00\x00'	bad	IP field failure: unexpected source port
b'gVh8VjgjAABNT0hiU0VOLVDAG9sb3JTaf1cGAoA'	Good	OK
b'\x9c\xc3\x81\x80\x00\x01\x00\x01\x00\x00\x00\x00\x00\x0cgstatica dssl\x01l\x06google\x03com\x00\x00\x1c\x00\x01\xc0\x0c\x00\x1c \x00\x01\x00\x00\x00w\x00\x10*\x00\x14P@\x02\x08\x07\x00\x0 0\x00\x00\x00\x00 \x03'	Good	OK

Example of ICMP dataset

Table 67 ICMP Dataset Part 1

Time Of Packet	version	Internet Header Length	Type Of Service	Total Length
01:06.3	4	20	Routine Normal delay Normal throughput Normal Reliability Normal monetary cost	69
39:36.6	4	20	Routine Normal delay Normal throughput Normal Reliability Normal monetary cost	270

Table 68 ICMP Dataset Part 2

Identification	Flags	Fragment Offset	Time To Live	Protocol	Check Sum
0x6a4d	0 - Reserved bit 0 - Fragment if necessary 0 - last fragment	0	128	Internet Control Message Protocol	0
0x7ecc	0 - Reserved bit 0 - Fragment if necessary 0 - last fragment	0	128	Internet Control Message Protocol	0

Table 69 ICMP Dataset Part 3

Src. Add Oct1	Src. Add Oct2	Src. Add Oct3	Src. Add Oct4	Dst. Add Oct1	Dst. Add Oct2	Dst. Add Oct3	Dst. Add Oct4
192	168	1	2	192	168	1	2
192	168	1	18	192	168	1	18

Table 70 ICMP Dataset Part 4

Type ICMP	Code ICMP	Check Sum ICMP	payload	Signature	Reason Of Signature Status
Destinati on unreacha ble.	1	36253	b'\x00\x00\x00\x00E\x00\x00)jL@\x00\x80\x06\x00\x00\xc0\xa8\x01\x024UEL\xe5\x00P\x18;\x95\xb4d\xb4\xfc\x84P\x10@\xcf>\xc7\x00\x00\x00'	Good	OK
Destinati on unreacha ble.	1	48040	b'\x00\x00\x00\x00E\x00\x00\xf2<A@\x00\x80\x06\x00\x00\xc0\xa8\x01\x12(M\xe5R\x19V\x01\xbb.\xaa\x10\x12\xf8\$=\xc3P\x19\x01\x00\x87e\x00\x00\x17\x03\x03\x00p\xfb\xfb0\xf4y"=\x86\x1d%d\x00/\x9e9X\xa3\xc8\xa4\xc3\xad\xe4J\x9a(\x05\xa6q\xd6\x8b\x95?KV\xb4\x1d\x19\xf6\xd1\x947\xda9\xc4\xcd\x91d\xb7;\x07\x80M`\xd4D\xd2tz#\x82\xae bE\xf6Sk\xd8\x8e\x8e=\xfd\xaa\xb3\x93\xc6\x15pD\x94\xb1R\xb8a\x9b\xff\xe1\xee9\x9a\xf1@\xb7\xd7\xd5y\xa6\xa5Tq\x828\xa0>\' \xfd\xe3\x90\xcb\x03H\x10\x15\x03\x03\x00PB\xeeG!N\xd1DEm\xc8pt\xef\r\x8e\x87)T\xa a\x07\xb2~=4\x91\r\xc5\xc1,\xfcX\xd5\xe0\x	Good	OK

Example of any other protocol dataset

Table 71 any protocol Dataset Part 1

Time Of Packet	version	Internet Header Length	Type Of Service	Total Length
11:33.6	4	24	Routine Normal delay Normal throughput Normal Reliability Normal monetary cost	40
11:47.1	4	24	Routine Normal delay Normal throughput Normal Reliability Normal monetary cost	40

Table 72 any protocol Dataset Part 2

Identification	Flags	Fragment Offset	Time To Live	Protocol	Check Sum
0x185b	0 - Reserved bit 0 - Fragment if necessary 0 - last fragment	0	1	IGMP for user Authentication Protocol. Internet Group Management Protocol. Router-port Group Management Protocol.	35508
0x1889	0 - Reserved bit 0 - Fragment if necessary 0 - last fragment	0	1	IGMP for user Authentication Protocol. Internet Group Management Protocol. Router-port Group Management Protocol.	35462

Table 73 any protocol Dataset Part 3

Src. Add Oct1	Src. Add Oct2	Src. Add Oct3	Src. Add Oct4	Dst. Add Oct1	Dst. Add Oct2	Dst. Add Oct3	Dst. Add Oct4
192	168	225	1	224	0	0	22
192	168	225	1	224	0	0	22

Table 74 any protocol Dataset Part 4

payload	signature	Reason Of Signature Status
b""\x00\xfa\x01\x00\x00\x00\x01 \x03\x00\x00\x00\xe0\x00\x00\x fc'	Good	OK
b""\x00\xfa\x01\x00\x00\x00\x01 \x04\x00\x00\x00\xe0\x00\x00\x fc'	Good	OK

6.2.4 Analyzing Snort Signature based system

Snort Comparison

Introduction
Because of the amount of information available, wide deployment, and open source nature of the Snort community, there has been an ongoing need to convert Snort signatures to be used with Cisco IPS. This section provides an overview of the conversion process.

Users are advised that many of the signatures widely available on the Internet have been developed by administrators who have limited knowledge of IPS signature development. Therefore, these signatures may detect a particular exploit instead of detecting all exploits. They may also consume high resources and need to be continually updated to keep pace with changing exploits. Such signatures should be used with caution.

The following figure is an example of a Snort rule:

Figure 13. Snort Rule Example

A Snort signature consists of the following sections:

Rule actions: The rule action tells Snort what to do when it finds a packet that matches the rule criteria. There are five available actions in Snort: alert, log, pass, activate, and dynamic. In inline mode, additional options such as drop, reject, and adrop are available.

Protocols: Snort supports the following protocols: TCP, UDP, ICMP, and IP.

Directional operator: The directional operator is used to indicate the direction of the traffic to which the Snort rule applies.

Rule options: The following table illustrates Snort rule options and their equivalent Cisco IPS signature parameters.

Figure 18 Snort rule form [7]

This figure shows part of snort rule:

Signature is the value in the content field. After we unpack packet and unpack the Payload we search for the signature in Payload.

If signature is found then there is a malicious behavior and this behavior is classified according to security experts' knowledge.

Note: signature is a static predefined content from malicious studied packets by security experts.

Example

```
alert tcp $HOME_NET any -> $EXTERNAL_NET $HTTP_PORTS (msg:"BLACKLIST User-Agent known malicious User-Agent string Baby Remote - Win32/Babmote.A"; flow:to_server ,established; content:"User-Agent|3A| Baby Remote";reference:url,www.virustotal.com/latest-report.html?resource=0712178d245f4e5a5d0cf6318bf39144; class type: Trojan - activity; sid:20009; rev:3;)
```

Parameters

alert: Role action.

tcp: Protocol.

\$HOME_NET: variable declared in snort configuration file for internal network IPs. It is the field for source address.

Any: is defined for field of source port.

->: Directional operator.

\$EXTERNAL_NET: variable declared in snort configuration file for external network IPs. It is the field for destination address.

\$HTTP_PORTS: variable declared in snort configuration file for HTTP ports. It is the field for destination port.

msg: message to user when alert occurs.

Flow: show direction of flow.

Content: field for signature.

Reference: contain url for the reference snort use.

Class type: contain name of class attack classified under it.

sid: field for signature ID.

rev: revision times of rule.

6.3 Model Planning

6.3.1 Techniques and Workflows and Model Selection

- Data is changed from unstructured data to structured data csv file format.
- Categorization using association rules and grouping into clusters.
- Text analysis for Document Representation and regular expression.

We started the program. We are collecting network traffic. We have now features of packet. We will use these features to classify network traffic. There are a lot of machine learning modules used for anomaly detection method. But, they use normal network traffic to implement the module. This traffic can contain malicious packets. They analyze the packets to get certain values for a feature and analyze bit stream.

We created a module that analyzes network traffic according to **network standards** such as RFC for IETF and IANA not only analyzing network traffic that depend on user behavior. This module is more generic. As, standard is more efficient to implement a module with zero error depending on zero entropy or knowledge accuracy. Knowledge accuracy is a measure describing knowledge. Knowledge using standards and how many of standards did you use and how did you integrate them. Integrating all RFCs together needs a lot of time. But, it is very efficient.

- Classification using Naive Bayesian then Decision Tree using new search tree OFDS (Only First Depth Search).

6.4 Model Building

6.4.1 Categorization

- Discover "interesting" relationships among fields of packet.
- Association rule for defining which cluster contains the packet is an assertion for protocol field.
- Categorize IPv4 packet to 4 groups each group in a cluster.
- Search in IP header for protocol field and according to protocol field the packet join its cluster.
- There are 4 clusters.
 - Cluster for TCP packets.
 - Cluster for UDP packets.
 - Cluster for ICMP packets.
 - Cluster for any other protocol packets.
- Association rules are added for malicious packets where there is signature. Packet has a column in dataset stands for signature status and signature class. Signature status is "Good" or "bad". Signature class is defined from the association rules to detect known attacks.

6.4.2 Text analysis techniques

- Partitioning packet fields according to standard structure for document representation using **struct** python module and splitting packet fields approach described in 6.2.2.
- Using regular expression to match the signature in the payload of packet to find malicious packets.

6.4.3 Classification

6.4.3.1 Brief Discussion of algorithms

6.4.3.1.1 Naive Bayesian Classifier

Used if you want to assign (known) labels to objects. Used to answer “Where in the dataset should I place this packet? Is this packet malicious? Labels in classifiers are pre-determined unlike in clustering we discover the structure and assign labels.

The Naïve Bayesian Classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong naïve independence assumptions. Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. It is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.

For some types of probability models, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without accepting Bayesian probability or using any Bayesian methods.

Despite their naive design and apparently oversimplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations. An advantage of naive Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification

Bayes' Law states: $P(B | A) * P(A) = P(A | B) * P(B)$.

That is, the probability that B is true given that A is true, times the probability of A is the same as the probability that A is true given that B is true, times the probability of B.

For an attribute A we have m classes a_1, a_2, \dots, a_m

The probability of A given a set of j values of b is the product of the conditional probability of every a_i given b_j

$$\begin{aligned} P(A|b_j) &= P(a_1, a_2, \dots, a_m | b_j) \\ &= \prod_i^m P(a_i | b_j) \end{aligned}$$

6.4.3.1.2 Decision Tree Classifier

Decision tree learning uses a decision tree as a predictive model which maps observations about an item (represented in the branches) to conclusions about the item's target value (represented in the leaves). It is one of the predictive modeling approaches used in statistics, data mining and machine learning. The decision tree classifiers organized a series of test questions and conditions in a tree structure. The following figure [7.10] shows an example decision tree for predicting whether the person cheats. In the decision tree, the root and internal nodes contain attribute test conditions to separate records that have different characteristics. All the terminal node is assigned a class label Yes or No. Tree models where the target variable can take a finite set of values are called classification trees; in these tree structures, leaves represent class labels and branches represent conjunctions of features that lead to those class labels. Decision trees where the target variable can take continuous values (typically real numbers) are called regression trees.

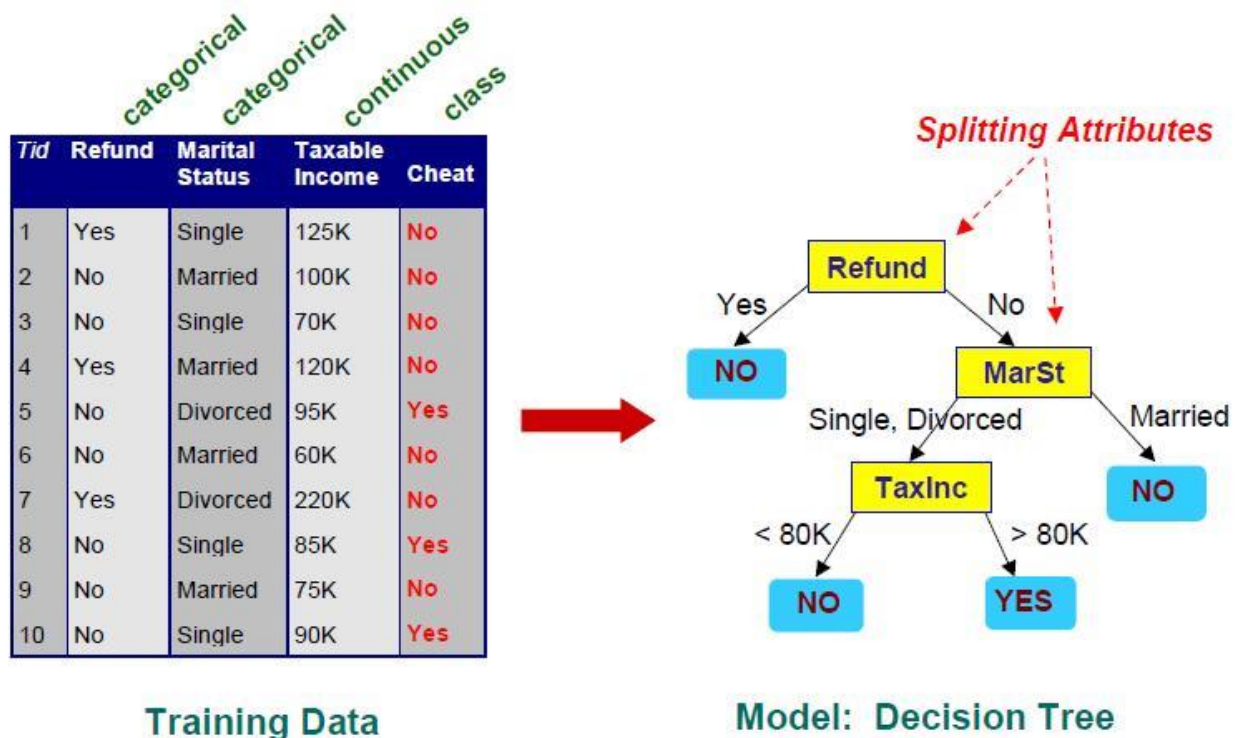


Figure 19 Example of Training Data and Decision tree model

In decision analysis, a decision tree can be used to visually and explicitly represent decisions and decision making. A decision tree describes data (but the resulting classification tree can be an input for decision making). The goal is to create a model that predicts the value of a target variable based on several input variables. An example is shown in Figure . Each interior node corresponds to one of the input variables; there are edges to children for each of the possible values of that input variable. Each leaf represents a value of the target variable given the values of the input variables represented by the path from the root to the leaf.

A decision tree is a simple representation for classifying examples. For this section, assume that all of the input features have finite discrete domains, and there is a single target feature called the classification. Each element of the domain of the classification is called a class. A decision tree or a classification tree is a tree in which each internal (non-leaf) node is labeled with an input feature. The arcs coming from a node labeled with an input feature are labeled with each of the possible values of the target or output feature or the arc leads to a subordinate decision node on a different input feature. Each leaf of the tree is labeled with a class or a probability distribution over the classes. A tree can be "learned" by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner called recursive partitioning. The recursion is completed when the subset at a node has all the same value of the target variable, or when splitting no longer adds value to the predictions. This process of *top-down induction of decision trees* (TDIDT) is an example of a greedy algorithm, and it is by far the most common strategy for learning decision trees from data. Decision trees can be described also as the combination of mathematical and computational techniques to aid the description, categorization and generalization of a given set of data.

We will implement a decision tree to classify the packets to good if it obeys documentation of standard and bad otherwise. No need to bother for the opposite direction which depends on checking if bad packet exists. So, there are many branches for a tree and many sub trees for classification.

We will also use a new search algorithm to span the tree. We will use OFDS (Only First Depth Search). We will design a tree as the figure [s]. For each node check you either continue in the right side or you drop the packet if the packets don't follow the expected standard. We are using data analytics platform. So, scalability exists and there is pipelining if the fields are independent.

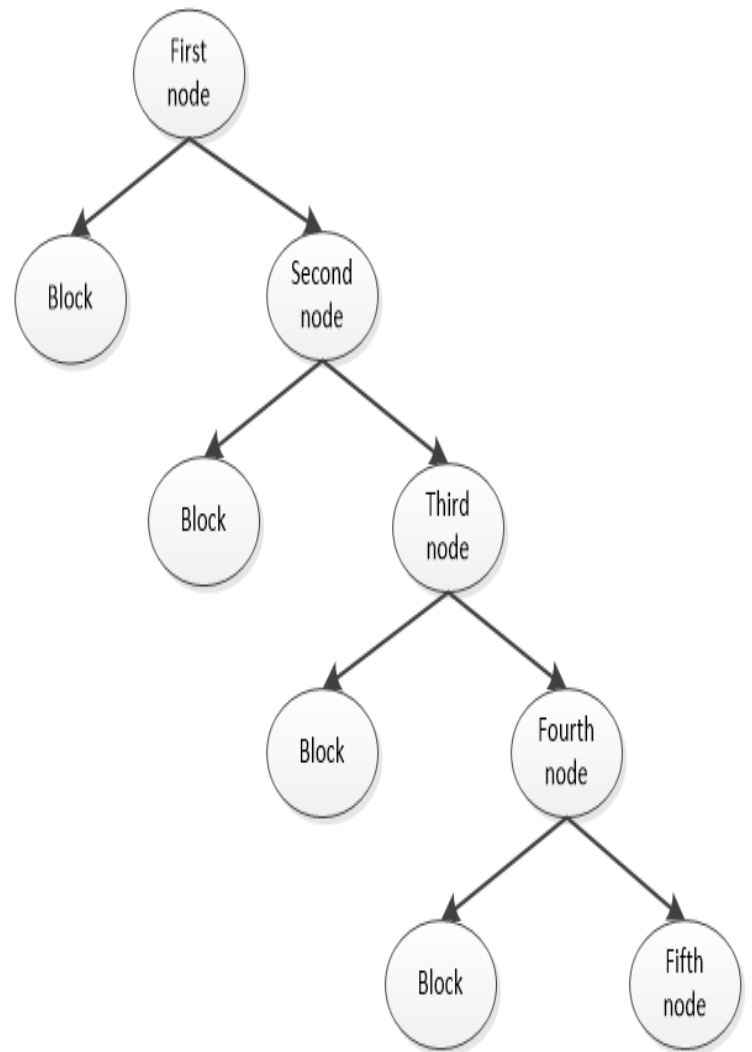


Figure 20 OFDS Decision tree model

6.4.3.1.3 Anomaly Detection Classifier

What we are trying to prove here is that in this case of study the anomaly detection that can be described as a Gaussian curve can be changed to a pulse and give same output.

The x-axis represents the time in decades.

The y-axis represents network traffic obeying standards.

Note: negative time has no meaning.

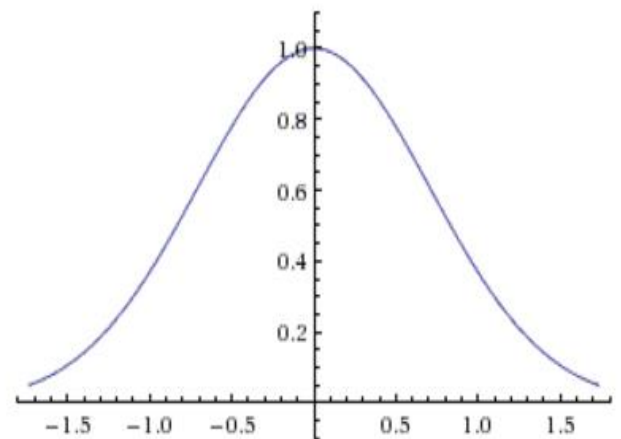


Figure 21 Gaussian curve [8]

During many years hackers have developed many attacks that are different from the standard communication protocols. We can detect them by checking if the connection and communication protocol is following the standards. But, there are few attacks that fellow standards which seem to be normal like DDoS (Distributed Denial of Service) attack. DDoS overwhelms computing resources for online servers. It seems really normal. Users are requesting service and online server responsible for service can't afford it. There are normal users and unexpected users.

Question:

If you get our point of view how could you solve DDoS attacks?

6.4.3.2 Use of Classifier

6.4.3.2.1 Naive Bayesian

In our dataset all attributes except signature status and reason for signature status are features for the label signature state.

Probability for a feature is classified into 2 states: probability of success = 1, probability of failure = 0.

Assume B stands for "Good Packet". Assume A stands for "No malicious signature".

Bayes' Law states: $P(B | A) * P(A) = P(A | B) * P(B)$.

That is, the probability that "Good Packet" is true given that "No malicious signature" is true, times the probability of "No malicious signature" is the same as the probability that "No malicious signature" is true given that "Good Packet" is true, times the probability of "Good Packet".

Probability that "No malicious signature" is true given that "Good Packet" is true is always true. So Bayes' law can be written as $P(B | A) * P(A) = P(B)$.

So, if $P(A) = 1$ which means probability of "No malicious signature" = 1 and if $P(A) = 1$ then $P(B|A) = 1$. Since $P(A)$ is dependent on $P(B|A)$ then Bayes' law is simplified to $P(B) = P(A)$

$P(\text{Good Packet}) = P(\text{No malicious signature})$

Is there a problem here?

You can't nowadays collect all malicious signatures. But, you can collect all standards behavior.

How about

$P(\text{Good Packet}) = P(\text{packet obeying standards})$

6.4.3.2.1.1 Naive Bayesian with Signature Based System

We concluded that probability to have a good packet depend on probability of not finding malicious signature. In signature based system approach for IPS, we ensure that there is no signature match in packet payload using regular expression. If there is malicious signature in payload this defect is categorized according to its family. Attribute “Reason of Signature Status” specifies family of attack in the dataset.

6.4.3.2.1.2 Naive Bayesian with Anomaly Based System

Probability of right packet fields is the dominant case. In anomaly based system approach for IPS, we ensure that everything is what it is supposed to be according to standards made by IETF and RFC. What is abnormal is not normal. So, if you defined the normal behavior anything different is abnormal. This goal is hard to achieve. As, developers must be always be up to date with latest IETF meetings and changes in RFC and network structure and standards. Because, if they missed adding a new protocol then this protocol is abnormal and will be dropped and IPS will be restricted to product last update.

Note: This project is up to date 6/26/2017 you will find the date of latest update in the front page. Don’t use this project if 3 months passed without updating the IPS with the latest network security changes.

6.4.3.2.2 Decision Tree Algorithm

We have discussed before that will implement a new algorithm to span the tree. So, which node comes first? Why?

Answer for which node comes first is to choose the node that gives an end for the tree either “yes” or “no” and the reason for that is time efficiency to reach a goal.

First, let’s discuss how data scientists prefer to span decision tree?

Most common way to choose the root of the tree and the place of each node is to check the entropy of each node and choose the one with the least entropy. Entropy is the measure of randomness. So, you can classify features value to a certain class in less time.

Here we will choose our entropy from the train data and then test the data again and compare accuracy. Each node is a check for packet validation and there are a lot of nodes. So, it is better to choose the worst node known that attackers use it most.

If there are many nodes to check then you may think this cost time! No, never mind time with big data platform. In cloud community data bricks scalability is done automatically if the data set is very large and pipelining takes place in each independent action. So, the tree depth is only “One” 😊😊.

We collected 56803 entries

Probability of good packet = $54558/56803 = 0.96$

Probability of bad packet (unexpected window size) = $919/56803 = 0.016$

Probability of bad packet (unexpected port number) = $1236/56803 = 0.02$

Probability of bad packet (Different signature matches) = $90/56803 = 0.00158$

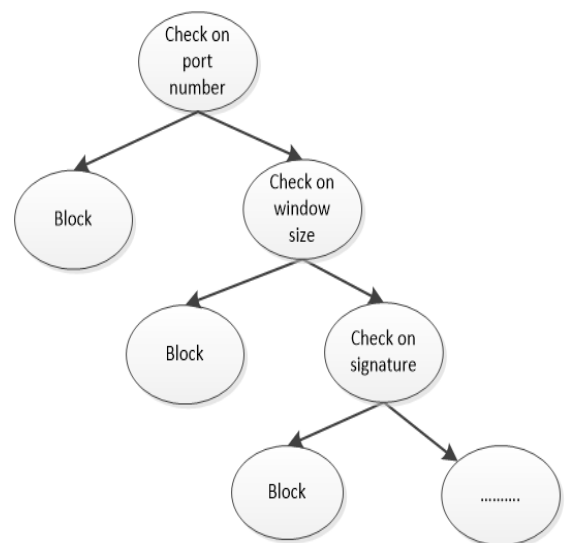


Figure 22 Example on OFDS Decision tree model

6.4.4 Accessing firewall and adding needed rules

We are going to access windows firewall to add rules needed to block malicious packets.

Netsh advfirewall firewall

In the netsh advfirewall firewall context, the add command only has one variation, the add rule command.

add rule:

Adds a new inbound or outbound firewall rule that filters traffic by allowing or blocking network packets that match the specified criteria.

Syntax

add rule

name = *RuleName*

dir = { **in** | **out** }

action = { **allow** | **block** | **bypass** }

[**program** = *ProgramPath\FileName*]

[**service** = { *ServiceShortName* | **any** }]

[**description** = *RuleDescription*]

[**enable** = { **yes** | **no** }]

[**profile** = { **public** | **private** | **domain** | **any** | [,...] }]

[**localip** = { *Addresses* }]

[**remoteip** = { *Addresses* }]

[**localport** = { **any** | *Integer* | **rpc** | **rpc-epmap** | **iphttps** | **teredo** | [,...] }]

[**remoteport** = { **any** | *Integer* | [,...] }]

[**protocol** = { **any** | *Integer* | **icmpv4** | **icmpv6** | **icmpv4:type,code** | **icmpv6:type,code** | **tcp** | **udp** }]

[**interfacetype** = { **any** | **wireless** | **lan** | **ras** }]

[**rmtcomputergrp** = *SDDLString*]

[**rmtusrgrp** = *SDDLString*]

[**edge** = { **yes** | **deferapp** | **deferuser** | **no** }]

[**security** = { **authenticate** | **authenc** | **authdynenc** | **authnoencap** | **notrequired** }]

Parameters

name = *RuleName*

Required. Specifies the name of this firewall rule. The name should be unique, and must not be "all".

dir = { in | out }

Required. Specifies whether this rule matches inbound or outbound network traffic.

In means inbound which means to get in you must pass some checks. As, to enter a film in cinema you must have a ticket. While, out means outbound which means to get out you must pass some checks. As, if you in a café and some said there is a thief then to get out you must not be the thief.

dir can be any of the following values:

in. The rule matches only inbound network traffic that is arriving at the computer. This rule appears in the Windows Firewall with Advanced Security MMC snap-in under **Inbound Rules**.

out. The rule matches only outbound network traffic that is sent by the computer. This rule appears in the Windows Firewall with Advanced Security MMC snap-in under **Outbound Rules**.

action = { allow | block | bypass }

Required. Specifies what Windows Firewall with Advanced Security does to filter network packets that match the criteria specified in this rule.

action can be one of the following:

allow. Network packets that match all criteria specified in this rule are permitted through the firewall.

block. Network packets that match all criteria specified in this rule are dropped by the firewall.

bypass. If **dir=in**, then this option is valid only for rules that have one or more accounts listed in **rmtcomputergrp** and optionally **rmtusrgrp**. Network packets that match this rule and that are successfully authenticated against a computer account specified in **rmtcomputergrp** and against a user account identified in **rmtusrgrp** are permitted through the firewall. If you specify this option, then you cannot

set **security=notrequired**. This option is the equivalent to the **Override block rules** checkbox in the Windows Firewall with Advanced Security MMC snap-in.

For computers that are running Windows 7 or Windows

Server 2008 R2, **action=bypass** is permitted on an outbound rule. Selecting this option on an outbound rule causes matching traffic to be permitted though this rule even if other matching rules would block the traffic. No accounts are required in **rmtcomputergrp** or **rmtusergrp** for an outbound bypass rule, however, if authorized or excepted computers are listed in those groups they will be enforced.

The **action=bypass** option on an outbound rule is not valid on computers that are running earlier versions of Windows.

[program = *ProgramPath\FileName*]

Specifies that network traffic generated by the identified executable program matches this rule.

If **program** is not specified, then network traffic generated by any program matches this rule.

[service = { *ServiceShortName* | any }]

Specifies that traffic generated by the identified service matches this rule.

The *ServiceShortName* for a service can be found in Services MMC snap-in, by right-clicking the service, selecting **Properties**, and examining **Service Name**.

If **service** is not specified then network traffic generated by any program or service matches this rule.

[description = *RuleDescription*]

Provides information about the firewall rule.

[enable = { yes | no }]

Specifies whether the rule is currently enabled.

If **enable** is not specified, the default is **yes**.

[profile = { public | private | domain | any | [,...] }]

Specifies the profile(s) to which the firewall rule is assigned. The rule is active on the local computer only when the specified profile is currently active.

You can include multiple entries for **profile** by separating them with a comma. Do not include any spaces.

If **profile** is not specified, the default is **any**.

[**localip** = { *Addresses* }]

Specifies that network packets with matching IP addresses match this rule. **localip** is compared to the Destination IP address field of an inbound network packet. It is compared to the Source IP address field of an outbound network packet.

localip can be any of the following values:

any. Matches any IP address.

IPAddress. Matches only the exact IPv4 or IPv6 address.

IPSubnet. Matches any IPv4 or IPv6 address that is part of the specified subnet. The format is the subnet address, followed by '/' and then either the number of bits in the subnet mask or the subnet mask itself.

IPRange. Matches any IPv4 or IPv6 addresses that fall within the specified range. The format is the starting and ending IP addresses of the range separated by a '-'.

Multiple entries can be specified for **localip** by separating them with a comma. Do not include any spaces.

If **localip** is not specified, the default is **any**.

[**remoteip** = { *Addresses* }]

Specifies that network packets with matching IP addresses match this rule. **remoteip** is compared to the Destination IP address field of an outbound network packet. It is compared to the Source IP address field of an inbound network packet.

remoteip can be any of the following values:

any. Matches any IP address.

localsubnet. Matches any IP address that is on the same IP subnet as the local computer.

dns|dhcp|wins|defaultgateway. Matches the IP address of any computer that is configured as the identified server type on the local computer.

IPAddress. Matches only the exact IPv4 or IPv6 address specified.

IPSubnet. Matches any an IPv4 or IPv6 subnet that is part of the specified subnet. The format is the subnet address, followed by '/' and then either the number of bits in the subnet mask or the subnet mask itself.

IPRange. Matches any IPv4 or IPv6 addresses that fall within the specified range. The format is the starting and ending IP addresses of the range separated by a '-'.

Multiple entries can be specified for **remoteip** by separating them with a comma.

If **remoteip** is not specified, the default is **any**.

[**localport** = { **any** | *Integer* | **rpc** | **rpc-epmap** | **teredo** | [,...] }]

Specifies that network packets with matching IP port numbers matched by this rule. **localport** is compared to the Source Port field of an outbound network packet. It is compared to the Destination Port field of an inbound network packet.

localport can be any of the following values:

any. Matches any value in the port field of the IP packet.

Integer. Specifies the exact port number that must be present for the packet to match the rule. The port values can be individual numbers from 0 through 65535

rpc. Matches inbound TCP packets that are addressed to the listening socket of an application that correctly registers the port as an RPC listening port. A rule with this option must also specify **protocol = tcp**, and **dir = in**. We recommend that you also specify the

appropriate **program = ProgramName** and/or **service = ServiceName** options to ensure that only the correct service can send or receive traffic by using this rule. This option eliminates the need to know the specific port numbers assigned to the application at when it starts.

rpc-epmap. Matches inbound TCP packets that are addressed to the dynamic RPC endpoint mapper service. A rule with this option must also specify **protocol = tcp**, and **dir = in**. We recommend that you also

specify **program = %windir%\system32\svchost.exe**, and **service = rpcss** to ensure that only the RPC service can send or receive network traffic by using this rule. This option eliminates the need to know the specific port numbers assigned to the service when it starts. If you have one or more rules that specify **localport = rpc**, then you must also create a rule with **localport = rpc-epmap** enabled. This allows both the incoming request to the mapper, and the subsequent packets to the ephemeral ports assigned by the RPC service.

Teredo. Matches inbound UDP packets that contain Teredo packets. Teredo is an IPv4 to IPv6 transition technology that allows IPv4 computers to communicate with IPv6 computers.

iphttps. Matches inbound TCP packets that contain HTTPS with embedded IPv6 packets. IP-HTTPS is a firewall traversal protocol that allows IPv6 packets that would otherwise be blocked if sent by using Teredo, 6to4, or native IPv6. HTTPS is almost universally permitted through a firewall, so IP over HTTPS is another mechanism that can be used when a firewall does not support other edge traversal protocols. The IP-HTTPS option is valid on computers that are running Windows 7 or Windows Server 2008 R2 only, and is ignored if applied by Group Policy to computers that are running earlier versions of Windows.

Multiple entries can be specified for **localport** by separating them with a comma. Do not include any spaces.

If **localport** is not specified, the default is **any**.

[remoteport = { any | *Integer* | [,...] }]

Specifies that network packets with matching IP port numbers match this rule. **remoteport** is compared to the Destination Port field of an outbound network packet. It is compared to the Source Port field of an inbound network packet.

remoteport can be any of the following values:

any. Matches any value in the port field of the IP packet.

Integer. Specifies the exact port number that must be present for the packet to match the rule. The port values can be individual numbers, a range.

Multiple entries can be specified for **remoteport** by separating them with a comma. Do not include any spaces.

If **remoteport** is not specified, the default is **any**.

[protocol = { any | *Integer* | icmpv4 | icmpv6 | icmpv4:type,code | icmpv6:type,code | tcp | udp }]

Specifies that network packets with a matching IP protocol match this rule.

protocol can be any of the following values:

any. Matches any value in the Protocol field of the IP packet.

Integer. Specifies the protocol by number that must be present for the packet to match the rule. The value can range from 0 through 255.

icmpv4. Specifies that all ICMP v4 packets match this rule.

icmpv6. Specifies that all ICMP v6 packets match this rule.

icmpv4:*type,code*. Specifies that only ICMP v4 network packets with the specified type and code match this rule. *type* and *code* can each be either the keyword **any**, or an integer ranging from 0 to 255.

icmpv6:*type,code*. Specifies that only ICMP v6 network packets with the specified type and code match this rule. *type* and *code* can each be either the keyword **any**, or an integer ranging from 0 to 255.

tcp. Specifies that only TCP traffic addressed to or from the ports identified by **localport** and **remoteport** matches this rule.

udp. Specifies that only UDP traffic addressed to or from the ports identified by **localport** and **remoteport** matches this rule.

Multiple entries can be specified for **protocol** by separating them with a comma. Do not include any spaces.

If **protocol** is not specified, the default is **any**.

[interfacetype = { any | wireless | lan | ras }]

Specifies that only network packets passing through the indicated interface types match this rule. Using this parameter allows you to specify different firewall requirements for each of the three main network types. The value must be one of the following:

any. Network packets passing through any of the interface types match this rule.

wireless. Network packets that pass through a wireless network adapter match this rule.

lan. Network packets that pass through a wired LAN adapter match this rule.

ras. Network packets that pass through a RAS interface, such as a VPN or dial-up network connection match this rule.

If **interfacetype** is not specified, the default is **any**.

[rmtcomputergrp = SDDLString]

Specifies that only network packets that are authenticated as coming from or going to a computer identified in the list of computer and group accounts match this rule.

If **rmtcomputergrp** is specified, then **security** must be set to either **authenticate** or **authenc**.

If **action=bypass**, then at least one computer or computer group account must be specified in **rmtcomputergrp**.

For **rmtcomputergrp** to match, the network traffic must be authenticated using a credential that carries computer account information.

[rmtusrgrp = *SDDLString*]

Specifies that only network packets that are authenticated as coming from or going to a user identified in the list of user and group accounts match this rule.

If **rmtusrgrp** is specified, then **security** must be set to either **authenticate** or **authenc**.

For **rmtusergrp** to match, the network traffic must be authenticated using a credential that carries user account information.

[edge = { yes | deferapp | deferuser | no }]

Valid only when **dir=in**. Specifies that traffic that traverses an edge device, such as a Network Address Translation (NAT) enabled router, between the local and remote computer matches this rule. The **deferapp** and **deferuser** options are valid on computers running Windows 7 and Windows Server 2008 R2 only. If set to **deferapp** or **deferuser**, then Windows allows the application or user to programmatically register with the firewall to receive inbound unsolicited application traffic from the edge device.

This option is the equivalent of the **Allow edge traversal** checkbox in the Windows Firewall with Advanced Security MMC snap-in.

If **edge** is not specified, the default is **no**.

[security = { authenticate | authenc | authdynenc | authnoencap | notrequired }]

Specifies that only network packets protected with the specified type of IPsec options match this rule.

security can be one of the following values:

authenticate. Network packets that are authenticated by IPsec match this rule. You must create a separate connection security rule to authenticate the traffic. This option is the equivalent of the **Allow only secure connections** in the Windows Firewall with Advanced Security MMC snap-in.

authenc. Network packets that are authenticated and encrypted by IPsec match this rule. You must create a separate connection security rule to authenticate and encrypt the traffic. This option is the equivalent of the **Require encryption** option in the Windows Firewall with Advanced Security MMC snap-in.

authdynenc. Network packets that are authenticated by IPsec match this rule, and if the initial packet is not already encrypted, then a new quick mode SA is negotiated with the remote host to encrypt the connection. All succeeding packets are authenticated and encrypted. If the negotiation of a quick mode encryption SA fails, then the firewall blocks the connection. You must create a separate connection security rule that requires authentication and encryption to permit negotiation of an appropriate encrypted SA. This option is the equivalent of the **Allow systems to dynamically negotiate encryption** in the Windows Firewall with Advanced Security MMC snap-in.

This option is valid for inbound firewall rules only.

authnoencap. Network connections that are authenticated, but not encapsulated by ESP or AH match this rule. This option is useful for connections that must be monitored by network equipment, such as intrusion detection systems (IDS), that are not compatible with ESP NULL-protected network packets. The initial connection is authenticated by IPsec by using AuthIP, but the quick mode SA permits cleartext traffic. To use this option, you must also configure a connection security rule that specifies **authnoencap** as a quick mode security method.

notrequired. Any network packet matches this rule, whether or not it is protected by IPsec. This option is the equivalent of not selecting the **Allow only secure connections** option in the Windows Firewall with Advanced Security MMC snap-in. If **security** is not specified, the default is **notrequired**. [9]

Example:

add rule name="block inbound tcp port 80" dir=in action=block localport=80 protocol=tcp

to check: show rule name="block inbound tcp port 80"

blocks inbound tcp traffic from port 80

```
netsh advfirewall firewall>add rule name="block inbound tcp port 80" dir=in action=block localport=80 protocol=tcp
Ok.

netsh advfirewall firewall>show rule name="block inbound tcp port 80"

Rule Name:                block inbound tcp port 80
-----
Enabled:                   Yes
Direction:                 In
Profiles:                  Domain,Private,Public
Grouping:
LocalIP:                   Any
RemoteIP:                  Any
Protocol:                  TCP
LocalPort:                 80
RemotePort:                Any
Edge traversal:             No
Action:                    Block
Ok.

netsh advfirewall firewall>delete rule name="block inbound tcp port 80"

Deleted 1 rule(s).
Ok.

netsh advfirewall firewall>show rule name="block inbound tcp port 80"

No rules match the specified criteria.

netsh advfirewall firewall>
```

Figure 23 Accessing Netsh AdvFirewall Firewall

6.5 Testing Tool Cloud Community Data Bricks

Databricks is a managed platform for running Apache Spark - that means that you do not have to learn complex cluster management concepts nor perform tedious maintenance tasks to take advantage of Apache Spark. Databricks also provides a host of features to help its users be more productive with Spark.

Apache Spark is a framework for large-scale data processing. Many traditional frameworks were designed to be run on a single computer. However, many datasets today are too large to be stored on a single computer, and even when a dataset can be stored on one computer (such as the datasets in this tutorial), the dataset can often be processed much more quickly using multiple computers.

Spark has efficient implementations of a number of transformations and actions that can be composed together to perform data processing and analysis. Spark excels at distributing these operations across a cluster while abstracting away many of the underlying implementation details. Spark has been designed with a focus on scalability and efficiency. With Spark you can begin developing your solution on your laptop, using a small dataset, and then use that same code to process terabytes or even petabytes across a distributed cluster.

In Spark, communication occurs between a driver and executors. The driver has Spark jobs that it needs to run and these jobs are split into tasks that are submitted to the executors for completion. The results from these tasks are delivered back to the driver.

The diagram shows an example cluster, where the slots allocated for an application are outlined in purple. (Note: We're using the term *slots* here to indicate threads available to perform parallel work for Spark.

At a high level, every Spark application consists of a driver program that launches various parallel operations on executor Java Virtual Machines (JVMs) running either in a cluster or locally on the same machine. In Databricks, "Databricks Shell" is the driver program. When running locally, pyspark is the driver program. In all cases, this driver program contains the main loop for the program and creates distributed datasets on the cluster, then applies operations (transformations & actions) to those datasets. Driver programs access Spark through a SparkContext object, which represents a connection to a computing cluster.

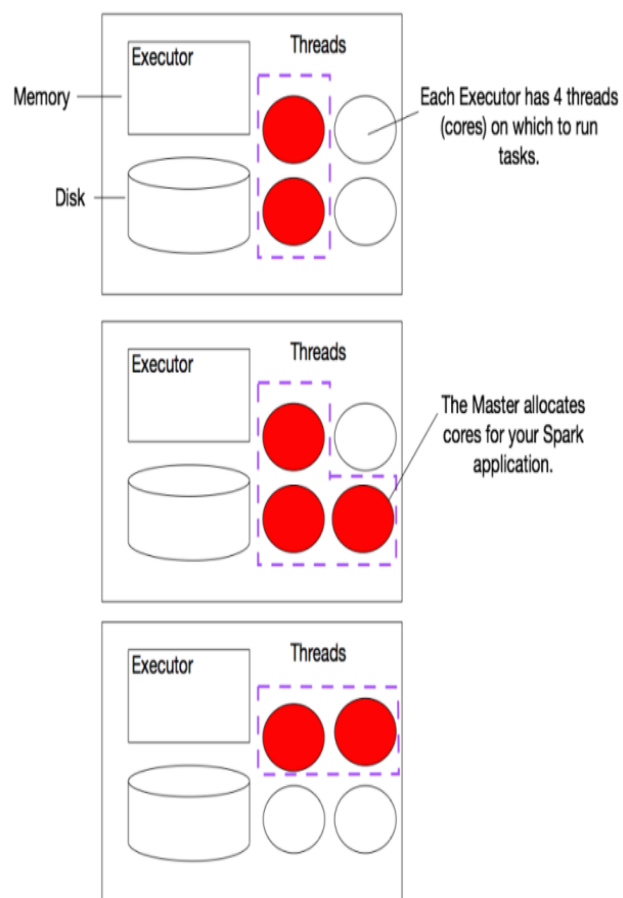


Figure 24 Example clusters [10]

In Spark, we first create a base Data Frame. We can then apply one or more transformations to that base Data Frame. A Data Frame is immutable, so once it is created, it cannot be changed. As a result, each transformation creates a new Data Frame. Finally, we can apply one or more actions to the Data Frames.

Note that Spark uses lazy evaluation, so transformations are not actually executed until an action occurs.

In Spark, datasets are represented as a list of entries, where the list is broken up into many different partitions that are each stored on a different machine. Each partition holds a unique subset of the entries in the list. Spark calls datasets that it stores "Resilient Distributed Datasets" (RDDs). Even Data Frames are ultimately represented as RDDs, with additional meta-data.

One of the defining features of Spark, compared to other data analytics frameworks (e.g., Hadoop), is that it stores data in memory rather than on disk. This allows Spark applications to run much more quickly, because they are not slowed down by needing to read data from disk. The figure to the right illustrates how Spark breaks a list of data entries into partitions that are each stored in memory on a worker.

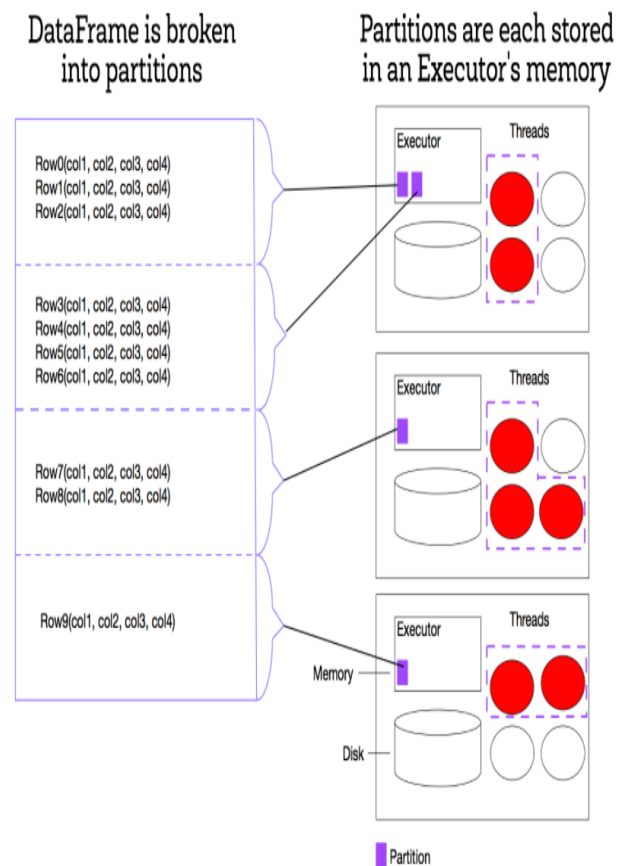


Figure 25 Partitioning data frame to "Resilient Distributed Datasets" (RDDs) [10]

A note about Data Frames and queries

When you use Data Frames, you are building up a *query plan*. Each transformation you apply to a Data Frame adds some information to the query plan. When you finally call an action, which triggers execution of your Spark job, several things happen:

1. Spark's Catalyst optimizer analyzes the query plan (called an *unoptimized logical query plan*) and attempts to optimize it. Optimizations include (but aren't limited to) rearranging and combining filter() operations for efficiency, converting Decimal operations to more efficient long integer operations, and pushing some operations down into the data source (e.g., a filter() operation might be translated to a SQL WHERE clause, if the data source is a traditional SQL RDBMS). The result of this optimization phase is an *optimized logical plan*.
2. Once Catalyst has an optimized logical plan, it then constructs multiple *physical* plans from it. Specifically, it implements the query in terms of lower level Spark RDD operations.
3. Catalyst chooses which physical plan to use via *cost optimization*. That is, it determines which physical plan is the most efficient (or least expensive), and uses that one.
4. Finally, once the physical RDD execution plan is established, Spark actually executes the job.

You can examine the query plan using the explain() function on a Data Frame. By default, explain() only shows you the final physical plan; however, if you pass it an argument of True, it will show you all phases.

Caching Data Frames

For efficiency Spark keeps your Data Frames in memory. (More formally, it keeps the RDDs that implement your Data Frames in memory.) By keeping the contents in memory, Spark can quickly access the data. However, memory is limited, so if you try to keep too many partitions in memory, Spark will automatically delete partitions from memory to make space for new ones. If you later refer to one of the deleted partitions, Spark will automatically recreate it for you, but that takes time.

So, if you plan to use a Data Frame more than once, then you should tell Spark to cache it. You can use the `cache()` operation to keep the Data Frame in memory. However, you must still trigger an action on the Data Frame, such as `collect()` or `count()` before the caching will occur. In other words, `cache()` is lazy: It merely tells Spark that the Data Frame should be cached when the data is materialized. You have to run an action to materialize the data; the Data Frame will be cached as a side effect. The next time you use the Data Frame, Spark will use the cached data, rather than precomputing the Data Frame from the original data.

6.6 Operationalize

Spark provides on demand scalability as we see in figure below that there are different datasets with different number of observations. Partitioning depend on the number of observations.

```
1 sparkpacket = sqlContext.read.format("csv").load("/FileStore/tables/8y2jlz11498565616889/PacketData.csv",header = True)
2 print "number of observation of different protocols is " + str(sparkpacket.count())
3 print "number of partitions is " + str(sparkpacket.rdd.getNumPartitions())
4 sparktcp = sqlContext.read.format("csv").load("/FileStore/tables/h3le299v1498568253180/TCPDataSet.csv",header = True)
5 print "number of observation of TCP is " + str(sparktcp.count())
6 print "number of partitions is " + str(sparktcp.rdd.getNumPartitions())
7 sparkudp = sqlContext.read.format("csv").load("/FileStore/tables/e7hi74yb1498576446020/UDPDataSet.csv",header = True)
8 print "number of observation of UDP is " + str(sparkudp.count())
9 print "number of partitions is " + str(sparkudp.rdd.getNumPartitions())
10 sparkicmp = sqlContext.read.format("csv").load("/FileStore/tables/swhzpo171498578194027/ICMPDataSet.csv",header = True)
11 print "number of observation of icmp is " + str(sparkicmp.count())
12 print "number of partitions is " + str(sparkicmp.rdd.getNumPartitions())
```

► (8) Spark Jobs

```
number of observation of different protocols is 352
number of partitions is 1
number of observation of TCP is 7204
number of partitions is 4
number of observation of UDP is 49126
number of partitions is 7
number of observation of icmp is 125
number of partitions is 1
```

Command took 1.82 seconds -- by abdullah.zakaria.94@gmail.com at 6/27/2017, 11:43:35 PM on final test

Figure 26 Spark scalability

We filtered our datasets using apache spark and get the same output.

7 User Guide

1-Python Setup

User must first setup python version 3 on his computer to be able to use our program and run commands on command prompt.

2-Using CMD

1-Open two CMD windows --> run as administrator

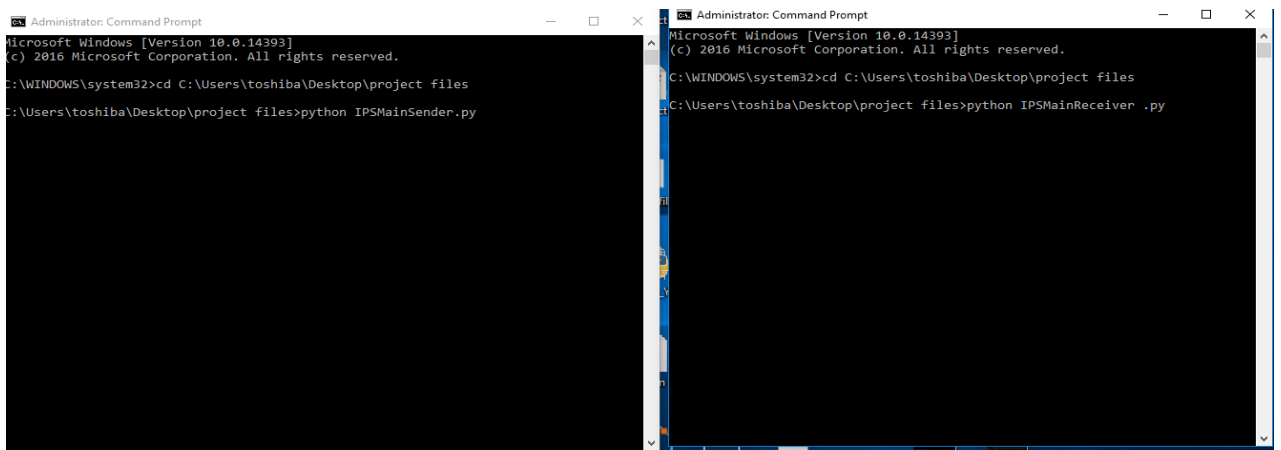


Figure 27 two windows

2-Go to the path of our program files:

cd path

Example:

cd C:/Users/toshiba/Desktop/project

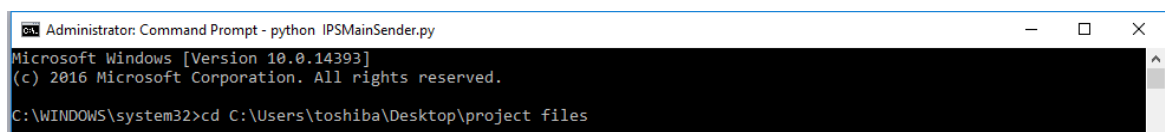
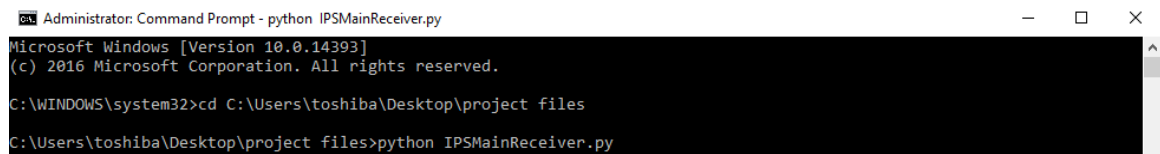


Figure 28 path example

3-Then type in the first window:

```
python IPSMainReceiver.py
```



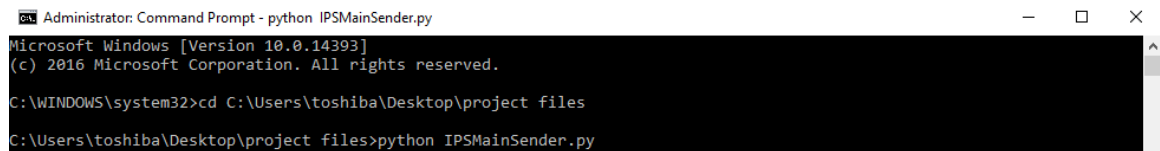
```
Administrator: Command Prompt - python IPSMainReceiver.py
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd C:\Users\toshiba\Desktop\project files
C:\Users\toshiba\Desktop\project files>python IPSMainReceiver.py
```

Figure 29 IPSMainReceiver

4-Then type in the second window:

```
python IPSMainSender.py
```



```
Administrator: Command Prompt - python IPSMainSender.py
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd C:\Users\toshiba\Desktop\project files
C:\Users\toshiba\Desktop\project files>python IPSMainSender.py
```

Figure 30 IPSMainSender

3-The program will run in background

The firewall will be turned off then the program will run in background sniffing the packets first the search in payload for bad signatures and the decision tree classifier will see if there are anomalies in the packets according to standards of RFC.

```
Administrator: Command Prompt - python IPSMainSender.py
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd C:\Users\toshiba\Desktop\project files
C:\Users\toshiba\Desktop\project files>python IPSMainSender.py
Ok.
Ok.
Ok.

An IP packet with the size 155 was Captured
Raw data:  b'\x00\x00\x9b7\x99\x00\x00\x04\x11\x00\x00\xa9\xfe\x90\x81\xef\xff\xff\xfa\xee\x05\x071\x00\x87\xb1\x97M-SEARCH * HTTP/1.1\r\nHost: 239.255.255.250:1900\r\nST: uuid:8563cbf0-2589-42de-8c69-b0779f2da115\r\nMan: "ssdp:discover"\r\nMX: 3\r\n\r\n'

Parsed data
Version          4
Header Length    20 Bytes
Type of Service  Routine
Normal Reliability
Length          155
Identification   14233
Flags            0 - Reserved bit
Fragment Offset  0
TTL             4
Protocol        User Datagram Protocol
Checksum        0
```

Figure 31 IPSMainSender running

```
Administrator: Command Prompt - python IPSMainReceiver.py
Microsoft Windows [Version 10.0.14393]
(c) 2016 Microsoft Corporation. All rights reserved.

C:\WINDOWS\system32>cd C:\Users\toshiba\Desktop\project files
C:\Users\toshiba\Desktop\project files>python IPSMainReceiver.py
Ok.
Ok.
Ok.

An IP packet with the size 78 was Captured
Raw data:  b'\x00\x00N\x032\x00\x00\x80\x11\x00\x00\xa9\xfe\x90\x81\xa9\xfe\xff\xff\x00\x89\x00\x89\x00:xu\xfd\xef\x01\x10\x00\x01\x00\x00\x00\x00\x00\x00 EEEFFDELFEFPACNFEE0EIFFDGEGDABM\x00\x00 \x00\x01'

Parsed data
Version          4
Header Length    20 Bytes
Type of Service  Routine
Normal Reliability
Length          78
Identification   0x332
Flags            0 - Reserved bit
Fragment Offset  0
TTL             128
Protocol        User Datagram Protocol
Checksum        0
Source          169.254.144.129
```

Figure 32 IPSMainReceiver running

4-Terminate the program

1-Enter CTRL+C on keyboard.

2-Type python Firewall.py.

To turn on the firewall before terminating the program

```
Identification          0x0
Flags                  0 - Reserved bit                1 - Do not fragment                0 - last fragment
Fragment Offset        0
TTL                    64
Protocol               User Datagram Protocol
Checksum              12886
Source                 4.4.4.1
Destination            255.255.255.255
Payload               b'\x16.\x16.\x00\x7f(\xef\x00\x00\xd1\xb6\x00\x01\x00\x06\xd4\xcam\xb2\xc6^\x00\x05\x00\x08MikroTik\x00\x07\x00\x046.27\x00\x08\x00\x08MikroTik\x00\n\x00\x04j{\x03\x00\x00\x0b\x00\thatatj-1WQF\x00\x0c\x00\x0cRB951Ui-2HnD\x00\x0e\x00\x01\x01\x00\x0f\x00\x10\xfe\x80\x00\x00\x00\x00\x00\x00\x0d\xcam\xff\xfe\xb2xc6^\x00\x10\x00\x07bridge1'
Source Port           5678
Destination Port      5678
Length of the UDP header 508
Checksum is          10479
Payload UDP           b'\x00\x00\xd1\xb6\x00\x01\x00\x06\xd4\xcam\xb2\xc6^\x00\x05\x00\x08MikroTik\x00\x07\x00\x046.27\x00\x08\x00\x08MikroTik\x00\n\x00\x04j{\x03\x00\x00\x0b\x00\thatatj-1WQF\x00\x0c\x00\x0cRB951Ui-2HnD\x00\x0e\x00\x01\x01\x00\x0f\x00\x10\xfe\x80\x00\x00\x00\x00\x00\x00\x0d\xcam\xff\xfe\xb2xc6^\x00\x10\x00\x07bridge1'

System is terminated

C:\Users\toshiba\Desktop\project files>
```

Figure 33 when pressing CTRL+C shows message System is terminated

```
Identification          0x0
Flags                  0 - Reserved bit                    1 - Do not fragment                0 - last fragment
Fragment Offset        0
TTL                   64
Protocol               User Datagram Protocol
Checksum              12886
Source                 4.4.4.1
Destination            255.255.255.255
Payload               b'\x16.\x16.\x00\x7f(\xef\x00\x00\xd\xb6\x00\x01\x00\x06\xd4\xcamlxb2\xc6^\x00\x05\x00\x08MikroTik\x00\x07\x00\x046.27\x00\x08\x00\x08MikroTik\x00\n\x00\x04j{\x03\x00\x00\x0b\x00\tHATJ-1WQF\x00\x0c\x00\x0cRB951Ui-2HnD\x00\x0e\x00\x01\x01\x00\x0f\x00\x10\xfe\x80\x00\x00\x00\x00\x00\x0d6xcaml\xff\xfe\xb2\xc6^\x00\x10\x00\x07bridge1'
Source Port           5678
Destination Port      5678
Length of the UDP header 508
Checksum is           10479
Payload UDP           b'\x00\x00\xd\xb6\x00\x01\x00\x06\xd4\xcamlxb2\xc6^\x00\x05\x00\x08MikroTik\x00\x07\x00\x046.27\x00\x08\x00\x08MikroTik\x00\n\x00\x04j{\x03\x00\x00\x0b\x00\tHATJ-1WQF\x00\x0c\x00\x0cRB951Ui-2HnD\x00\x0e\x00\x01\x01\x00\x0f\x00\x10\xfe\x80\x00\x00\x00\x00\x00\x00\x0d6xcaml\xff\xfe\xb2\xc6^\x00\x10\x00\x07bridge1'

System is terminated

C:\Users\toshiba\Desktop\project files>python Firewall.py
Ok.

Ok.

Ok.
```

Figure 34 when typing python Firewall.py

8 Output Results and Future Development

8.1 Overview

- 1) Analysis of the system.
- 2) Start of Design of the system.
- 3) Capturing Network traffic.
- 4) Sniffing packets and unpack the returned data according to segment structure defined before.
- 5) Analyzing IP header and get the type of service, flag, protocol, version, total Length, Identification, Internet Header Length, fragment Offset, Time to live, checksum, Source Address and destination Address. Detect Payload. Print data.
- 6) Analyzing TCP protocol according to protocol field analyzed in IP header and analyze their fields (source Port, destination Port, sequence Number, acknowledgment Number, data Offset, reserved, Notification, Congestion, Explicit, Control U, Control A, Control P, Control R, Control S, Control F, window, checksum, Urgent Pointer, Payload) and detect Payload and print data.
- 7) Analyzing UDP protocol according to protocol field analyzed in IP header and analyzes their fields (source Port, destination Port, Header length, checksum, Payload) and detect Payload and print data.
- 8) Analyzing ICMP protocol according to protocol field analyzed in IP header and analyzes their fields (Type, code, checksum, Payload) and detect Payload and print data.
- 9) We start to match packets' payload content with signatures to detect malicious packets using regular expressions in application layer part.
- 10) check packet fields' that they are following the standards.

8.2 Snippets of output

Figure 35 shows analyzing IP header and get the type of service, flag, protocol, version, total Length, Identification, Internet Header Length, fragment Offset, Time to live, checksum, Source Address and destination Address. Detect Payload. Print data. Analyzing TCP protocol according to protocol field analyzed in IP header and analyze their fields (source Port, destination Port, sequence Number, acknowledgment Number, data Offset, reserved, Notification, Congestion, Explicit, Control U, Control A, Control P, Control R, Control S, Control F, window, checksum, Urgent Pointer, Payload) and detect Payload and print data.

```
An IP packet with the size 52 was Captured
Raw data:  b'\x00\x0042a\x00\x00*\x06\xcc[J}\x85\xbc\x00\xa8\x01\x05\x141\xfcS\x03\x85*\x0e\x11\x04\xa8\x00\x10\x00\x00\x854\x00\x00\x01\x01\x05\n\x0e\x11\x04\xa7\x0e\x11\x04\xa8'
```

Parsed data					
Version	4				
Header Length	20 Bytes				
Type of Service	Routine	Normal delay	Normal throughput	Normal Reliability	Normal monetary cost
Length	52				
Identification	0x3261				
Flags	0 - Reserved bit	0 - Fragment if necessary	0 - last fragment		
Fragment Offset	0				
TTL	42				
Protocol	Transmission Control Protocol				
Checksum	52348				
Source	74.125.133.188				
Destination	192.168.1.5				
Payload	b'\x141\xfcS\x03\x85*\x0e\x11\x04\xa8\x00\x10\x00\x00\x854\x00\x01\x01\x05\n\x0e\x11\x04\xa7\x0e\x11\x04\xa8'				
Source Port	5228				
Destination Port	64616				
Sequence Number	1405322538				
Acknowledgment	3725718696				
Data Offset	32				
Reserved in TCP	0				
Notification bit (N, NS, Nonce Sum)	0				
Congestion bit (C, CWR)	0				
Explicit bit (E, ECE, ECN-Echo)	0				
Urgent pointer valid flag	0				
Acknowledgment number valid flag	1				
Push flag	0				
Reset connection flag	0				
Synchronize sequence numbers flag	0				
End of data flag	0				
The number of data bytes beginning with the one indicated in the acknowledgment field which the sender of this segment is willing to accept	176				
Checksum is	34100				
If the URG bit is set, this field points to the sequence number of the last byte in a sequence of urgent data.	0				
options TCP	b'\x01\x01\x05\n\x0e\x11\x04\xa7\x0e\x11\x04\xa8'				
Payload TCP	b''				

Figure 35 Example of sniffing packet and analyzing IP Header and TCP Header


```
An IP packet with the size 90 was Captured
Raw data:  b'\x00\x02\x00\x0e\x00\x00\x01\x00\x00\xc0\xa8\x01\x05\xc0\xa8\x01\x05\xd3\x01\x08\x10\x00\x00\x00\xe0\x00\x00>/\xda\x00\x00\x00\x11\x00\x00\xc0\xa8\x01\x05\xc0\xa8\x01\x01\xcc\xeb\x00*\xb6\xf1\xf8-\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x03dns\x08msftncsi\x03com\x00\x01\x00\x01'
Parsed data
Version      4
Header Length 20 Bytes
Type of Service Routine          Normal delay          Normal throughput          Normal Reliability          Normal monetary cost
Length       90
Identification 0xe
Flags         0 - Reserved bit          0 - Fragment if necessary          0 - last fragment
Fragment Offset 0
TTL           128
Protocol      Internet Control Message Protocol
Checksum      0
Source        192.168.1.5
Destination   192.168.1.5
Payload       b'\x03\x01\x10\x00\x00\x00\x00\xe0\x00>/\xda\x00\x00\x00\x11\x00\x00\xc0\xa8\x01\x05\xc0\xa8\x01\x01\xcc\xeb\x00*\xb6\xf1\xf8-\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x03dns\x08msftncsi\x03com\x00\x01\x00\x01'
Type of ICMP: Destination unreachable.
Code of ICMP: 1
ICMP header checksum: 2064
Payload ICMP  b'\x00\x00\x00\x00\x00>/\xda\x00\x00\x00\x11\x00\x00\xc0\xa8\x01\x05\xc0\xa8\x01\x01\xcc\xeb\x00*\xb6\xf1\xf8-\x01\x00\x00\x01\x00\x00\x00\x00\x00\x00\x00\x00\x00\x03dns\x08msftncsi\x03com\x00\x01\x00\x01'
An IP packet with the size 91 was Captured
Raw data:  b'\x00\x00[\x00\x0f\x00\x00\x00\x01\x00\x00\xc0\xa8\x01\x05\xc0\xa8\x01\x05\xd3\x01\xfb\x00\x00\x00\xe0\x00\x00?\x13\xef\x00\x00\x00\x11\x00\x00\xc0\xa8\x01\x05\x08\x08\x08\xe1\x005\x00+\xaa71\x01\x00\x00\x01\x00\x00\x00\x00\x00\x06config\x07nos-avg\x02cz\x00\x00\x10\x00\x01'
```

8.3 Project Future Development

First, we will apply computer forensic technology and topology to our project. Computer forensic or sometimes called digital forensic is our next phase. Digital forensic is a part of forensic science. It is concerned with collecting, preserving, analyzing and reporting evidence when incident occurs in an applicable format to be introduced to court. There are many tools that are many used forensic tools for bit imaging. Disk imaging differs from disk copy. When you use cp command in Linux to copy file you copy bits of files from the beginning of file to end of file marker. But, an image to a file copy the file till the end of file marker and the deleted part if exists. This deleted part is important as attacker after attacking a host they delete files used for attacks. So, disk image will get these files. Besides imaging doesn't change access time for file which is very important clue in cyber-crime. cp command in Linux change file access time which will destroy report honesty.

Second, we have discussed before that we are developing a module collecting all standards. We proposed to make this module a standard module to check any network traffic now and in future with same efficiency. This will take time and effort and module will be updated a lot during module implementation because standards have many dependencies. We have started SCA module and we will try to finish it in next year. There are various dependencies between packets. We will analyze series of packets and consider these dependencies.

References

- [1] Comparison between IPS and IDS,
<https://www.youtube.com/watch?v=zIMO-gp5hpl&feature=youtu.be&t=30>
- [2] "Software Development Life Cycle,"
https://www.tutorialspoint.com/software_engineering/software_development_lifecycle.htm
- [3] "Network Sorcery IP Protocol,"
<http://www.networksorcery.com/enp/protocol/ip.htm>
- [4] "Network Sorcery TCP Protocol,"
<http://www.networksorcery.com/enp/protocol/tcp.htm>
- [5] "Network Sorcery UDP Protocol,"
<http://www.networksorcery.com/enp/protocol/udp.htm>
- [6] "Network Sorcery ICMP Protocol,"
<http://www.networksorcery.com/enp/protocol/icmp.htm>
- [7] "Writing Custom Signatures for the Cisco Intrusion Prevention System – Cisco,"
<http://www.cisco.com/c/en/us/about/security-center/ips-custom-signatures.html#23>
- [8] Gaussian curve, <http://www.cut-the-knot.org/proofs/ClassicalIntegral.shtml>
- [9] Netsh AdvFirewall Firewall Commands
[https://technet.microsoft.com/sv-se/library/dd734783\(v=ws.10\).aspx](https://technet.microsoft.com/sv-se/library/dd734783(v=ws.10).aspx)
- [10] Spark Tutorial: Learning Apache Spark,
<https://community.cloud.databricks.com/?o=3677789241321331#notebook/1927471359023278/command/1927471359023281>
- [11] Internet Control Message Protocol (ICMP) Parameters,
<https://www.iana.org/assignments/icmp-parameters/icmp-parameters.xhtml>