

INTRODUCTION

The Internet is becoming increasingly dangerous; not only for large companies, but home networks are also being targeted by malicious activity. For hackers, well-traveled paths make networks more vulnerable than ever before and with relatively little expertise hackers have significantly impacted the networks of leading brands or government agencies. Cyber crime is also no longer the prerogative of lone hackers or random attackers. Today disgruntled employees, unethical corporations, even terrorist organizations all look to the Internet as a portal to gather sensitive data and instigate economic and political disruption. With networks more vulnerable and hackers equipped to cause havoc it's no surprise that network attacks are on the rise. According to a 2001 report, by Computer Security Institute (CSI) and the FBI, 70 percent of respondents acknowledged that their networks were attacked over the previous 12 months and 30 percent didn't know whether their networks were attacked but couldn't be sure! In addition, Denial of Service attacks increased by an astonishing 33 percent over the same period. And all this took place across networks, where firewalls had been installed in 90 percent of instances.

It's clear that we need security vendors to step up and deliver innovative solutions that effectively protect their networks from malicious attacks and misuse. As more devices in our are being connected to the Internet and more of our private data, such as photos, videos and financial information are stored digitally, our vulnerability against intrusions is increased. And if home network security is not taken seriously, we may become unnecessarily exposed to malicious users that seek to destroy or exploit our digital lifestyle. Therefore it is necessary to take a closer look at how home users can become better protected. Home users may not be aware of the dangers on the Internet and therefore cannot actively protect themselves against these dangers. In order to effectively be able to protect a network from intrusion the user needs to be aware of that the network is under attack. An Intrusion Detection System (IDS) is a system that monitors the network in real time and detects malicious activity. An IDS could therefore be used to help home users protect their network from malicious activity]. So why aren't more people taking advantage of this useful tool? We believe the reason is because an IDS is often company grade and not designed to be used in home networks, it may be too difficult for home users to implement it as a security solution, as well as be too expensive. Home users and small size companies could benefit from a small scale solution with a low implementation and maintenance cost.

One possible solution is to run an intrusion detection software, for example Snort, on a Raspberry Pi. The device is affordable and flexible as it can run a number of operating systems and might therefore be a very suitable device to provide an entry level upgrade in network protection. Thus introducing first ever real-time network intrusion prevention platform that takes Intrusion Detection Systems (IDS) to a new level. It is a cheap alternative to commercial IDS solutions, requires minimal computer skills by an end user to install and is a secure replacement to the home router without losing and functionality.

Information is a strategic resource. Organizations spend a significant amount of their budget on managing information resources. Computer security has several security related objectives among them the three fundamental objectives are: Secrecy i.e. to protect information; Incorruptibility, to protect information accuracy; lastly Access, to

ensure information delivery. It is necessary to put high priority to system security, minimize loop holes and secure the computer system against intrusion. Today's standard of security implement a configured firewall along with an intrusion detection system. If an intruder is able to acquire a weakness in the network by scanning the host network, he can easy penetrate into the system and obtain valuable data. If an intruder is masking his identity for a firewall enabled service, intrusion detection systems cannot minimize the damages. Let us study about the basic terminologies in network security: Intruder, Intrusion and Intrusion Detection System.

1.1 What is an Intrusion?

An intrusion is somebody attempting to break into or misuse your system. The word "misuse" is broad, and can reflect something severe as stealing confidential data to something minor such as misusing your email system for Spam.

1.2 Who is an Intruder?

Intruders can be external or internal and their intents may vary from serious to non-harmful. Statistically 80 percent of security breaches are committed by internal users and these are by far the most difficult to detect and prevent.

1.3 What is an IDS?

An Intrusion Detection System is the real-time monitoring of network/system activity. It is used for analyses of data for potential vulnerabilities and attacks in progress.



Fig 1.1: An NIDS Device

Intrusion Detection Systems is a topic that has recently garnered much interest in the computer security community. In the last few years, this interest level has spurred the development of a variety of approaches to providing IDS capabilities that are both reliable and low-impact in terms of management or cost. When presented with different

types of IDS one might be tempted to assume that one approach or another was inherently superior. In fact, the mixture of approaches used for IDS offers the security analyst a unique opportunity in terms of the synergies inherent in combined techniques. Intrusion Detection Systems are like a burglar alarm for your computer network. They detect unauthorized access attempts. They are the first line of defence for your computer systems.

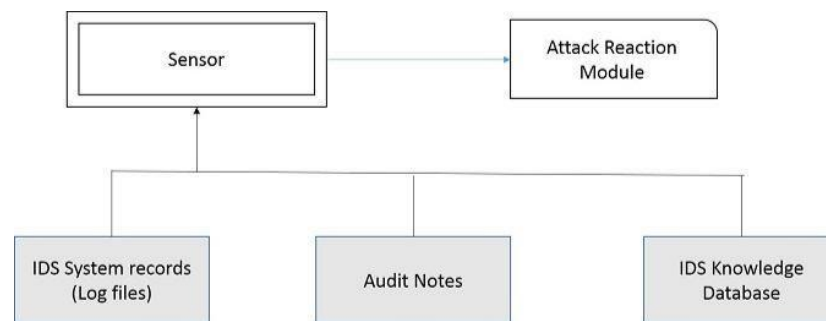


Fig 1.2: Intrusion Detection System

1.4 Types of IDS

1) Active & Passive IDS

An active Intrusion Detection Systems is also known as Intrusion Detection and Prevention System (IDPS). This Intrusion Detection and Prevention System is configured to automatically block suspected attacks without any intervention required by an operator. IDPS has the advantage of providing real-time corrective action in response to an attack.

A passive IDS is a system that's configured to only monitor and analyze network traffic activity and alert an operator to potential vulnerabilities and attacks. A passive IDS is not capable of performing any protective or corrective functions on its own.

2) Network based & Host based

A **Network Intrusion Detection system (NIDS)** transparently monitors network traffic, looking for patterns indicative of an attack on a computer or network device. By examining the network traffic, a network based intrusion detection system can detect suspicious activity such as a port scan or Denial of Service (DOS) attacks.

An NID monitors the network traffic it has access to, by comparing the data in the TCP/IP packet to a database of attack signatures. In a network environment, it can see packets to and from the system(s) that it monitors. In a switched environment, it can see packets coming to and from the system(s) that it monitors, providing it can see all

data traffic on the ports that connect to the systems. Once a NIDS detects an attack, the following actions may be taken:

- ✗ Send email notification
- ✗ Send an SNMP trap to a network management system
- ✗ Send a page (to a pager)
- ✗ Block a TCP connection
- ✗ Kill a TCP connection
- ✗ Run a user defined script

In general terms a NID will be deployed on a DMZ. This assumes that you have a firewall in place and that you have a DMZ configured. When deployed behind the firewall, the NID will detect attacks from protocols and sources allowed through the firewall and from internal users. By taking an action, such as sending an SNMP trap or a page, it can alert network staff that an attack is in progress and enable them to make decisions based on the nature of the attack. It is recommended that the IDS is used for detection and alerting only and not for proactive defence i.e. killing/blocking TCP connections as this can often cause more problems.

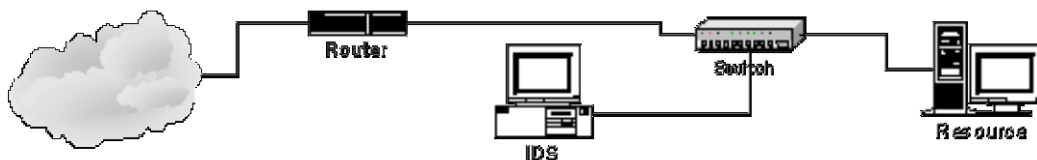


Fig 1.3: NIDS

In most cases, a **Host Intrusion Detection System (HIDS)** component is made up of two parts: a centralised manager and a server agent. The manager is used to administer and store policies, download policies to agents and store information received by agents. The agent is installed onto each server and registered with the manager. Agents use policies to detect and respond to specific events and attacks. An example of a policy would be an agent that sends an SNMP trap when three concurrent logins as root have failed on a UNIX server. System logs and processes are also monitored to see if any actions that violate the policy have occurred. If a policy has been violated, the agent will take a predefined action such as sending an email or sending a SNMP trap to a network management system. Host based intrusion detection system may further be divided into:

- **System integrity verifiers (SIV):** monitors system files to find when an intruder changes them (thereby leaving behind a backdoor). The most famous of such systems is "Tripwire". A SIV may watch other components as well, such as the Windows registry and configuration, in order to find well known signatures. Many existing products in this area should be considered more "tools"

than complete "systems": i.e. something like "Tripwire" detects changes in critical system components, but doesn't generate real-time alerts upon an intrusion.

- **Log file monitors (LFM):** monitor log files generated by network services. In a similar manner to NIDS, these systems look for patterns in the log files that suggest an intruder is attacking. A typical example would be a parser for HTTP server log files that looking for intruders who try well-known security holes, such as the "phf" attack. Example: swatch

3) Knowledge based & Behavior based

Each intrusion leaves a footprint behind (e.g., nature of data packets, failed attempt to run an application, failed logins, file and folder access etc.). These footprints are called signatures. A **Knowledge-based or Signature-based Intrusion Detection System** references a database of previous attack signatures and known system vulnerabilities. Based on these signatures Knowledge-based IDS identify intrusion attempts. The disadvantages of Signature-based Intrusion Detection Systems are that signature database must be continually updated and maintained and Signature-based Intrusion Detection Systems may fail to identify unique attacks.

A **Behavior-based or Anomaly-based Intrusion Detection System** references a baseline or learned pattern of normal system activity to identify active intrusion attempts. Deviations from this baseline or pattern cause an alarm to be triggered.

1.5 IDS vs. Firewall

A common misunderstanding is that firewalls recognize attacks and block them. This is not true. Firewalls are simply a device that shuts off everything, then turns back on only a few well-chosen items. In a perfect world, systems would already be "locked down" and secure, and firewalls would be unneeded. The reason we have firewalls is precisely because security holes are left open accidentally. Thus, when installing a firewall, the first thing it does is stops ALL communication. The firewall administrator then carefully adds "rules" that allow specific types of traffic to go through the firewall. For example, a typical corporate firewall allowing access to the Internet would stop all UDP and ICMP datagram traffic, stops incoming TCP connections, but allows outgoing TCP connections. This stops all incoming connections from Internet hackers, but still allows internal users to connect in the outgoing direction.

In summary, a firewall is not the dynamic defensive system that users imagine it to be. In contrast, an IDS is much more of that dynamic system. An IDS does recognize attacks against the network that firewalls are unable to see.

For example, in April of 1999, many sites were hacked via a bug in ColdFusion. These sites all had firewalls that restricted access only to the web server at port 80. However, it was the web server that was hacked. Thus, the firewall provided no defence. On the other

hand, an intrusion detection system would have discovered the attack, because it matched the signature configured in the system.

Another problem with firewalls is that they are only at the boundary to your network. Roughly 80% of all financial losses due to hacking come from inside the network. A firewall a perimeter of the network sees nothing going on inside; it only sees that traffic which passes between the internal network and the Internet.

Some reasons for adding IDS to your firewall are:

- ✗ Double-checks misconfigured firewalls.
- ✗ Catches attacks that firewalls legitimate allow through (such as attacks against web servers).
- ✗ Catches attempts that fail.

1.6 Why use IDS?

A correct firewall policy can minimize the exposure of many networks however they are quite useless against attacks launched from within. Hackers are also evolving their attacks and network subversion methods. These techniques include email based Trojan, stealth scanning techniques, malicious code and actual attacks, which bypass firewall policies by tunneling access over allowed protocols such as ICMP, HTTP, DNS, etc. Hackers are also very good at creating and releasing malware for the ever-growing list of application vulnerabilities to compromise the few services that are being let through by a firewall.

IDS arms your business against attacks by continuously monitoring network activity, ensuring all activity is normal. If IDS detects malicious activity it responds immediately by destroying the attacker's access and shutting down the attack. IDS reads network traffic and looks for patterns of attacks or signatures, if a signature is identified, IDS sends an alert to the Management Console and a response is immediately deployed.

The reason of developing this tool was the following:

- IDS(s) are difficult to setup
- Costs are high

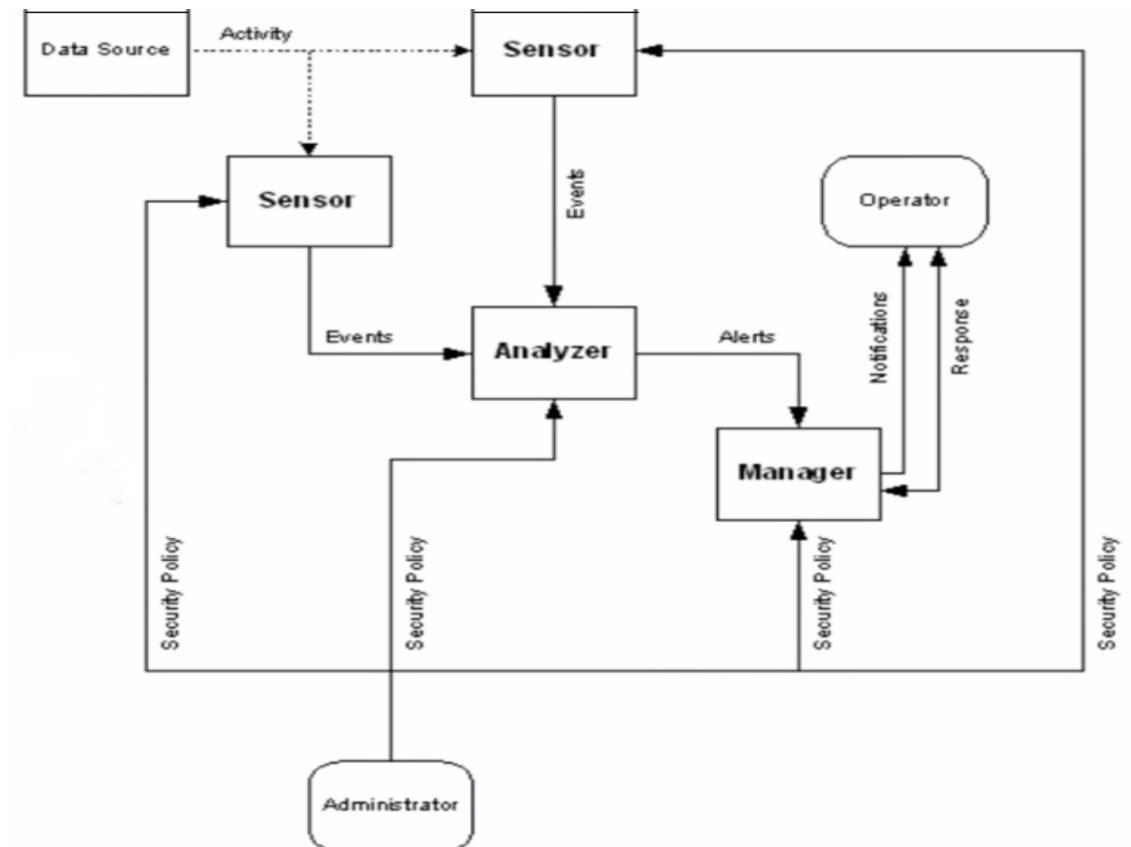


Fig 1.4: Basic IDS architecture

1.7 Functions of IDS

The IDS consist of four key functions namely, data collection, feature selection, analysis and action, which is given in Figure

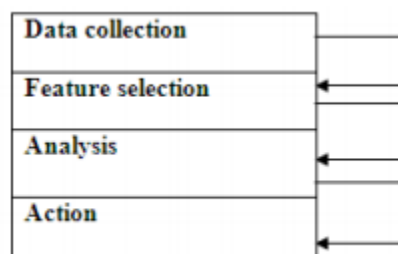


Fig 1.5: Functions of IDS

Data collection

This module passes the data as input to the IDS. The data is recorded into a file and then it is analyzed. Network based IDS collects and alters the data packets and in host based IDS collects details like usage of the disk and processes of the system.

Feature Selection

To select the particular feature large data is available in the network and they are usually evaluated for intrusion. For example, the Internet Protocol (IP) address of the source and target system, protocol type, header length and size could be taken as a key for intrusion.

Analysis

The data is analyzed to find the correctness. Rule based IDS analyze the data where the incoming traffic is checked against predefined signature or pattern. Another method is anomaly based IDS where the system behavior is studied and mathematical models are employed to it.

Action

It defines about the attack and reaction of the system. It can either inform the system administrator with all the required data through email/alarm icons or it can play an active part in the system by dropping packets so that it does not enter the system or close the ports.

1.8 IDS Life Cycle

Vendors frequently release new IDS products aggressively and compete for market shares. Estimating the new systems is not a relevant task and product calculation information is imperfect. Hiring and retaining the workers to administer security and intrusion detection are the challenging tasks. Faster changes in IT make it problematic for the firm to implement long term security strategy.

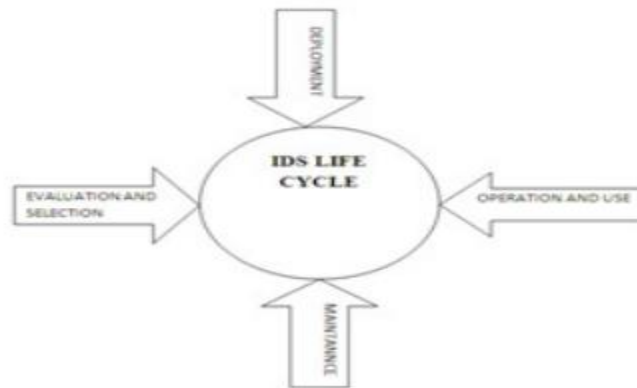


Fig 1.6: IDS Life Cycle

Evaluation and Selection

If an organization plans to get IDS it should examine the resources available for the systems operation and maintenance. Lifecycle of a product for economic IDS is accelerated. The third-party evaluation is available and their reports are generally on the surface. This process illustrates about the finding of the intruder and the amount of work required for maintaining the system in the network with traffic and the selection process defines about the identification of character, approaches, accuracy, usability, and effectiveness.

Deployment

Deployment phase includes the working of sensors to maximize protection for the critical assets by configuring the IDS to reflect security policy and installing signatures. Users must develop rules for handling the alerts and to associate alerts with other systems. The Intrusion Detection Working Group of the Internet Engineering Task Force (IETF) is developing common alert format that uses the IDS to alert from different systems and they are reported to a common display console.

Operation and use

Organization administers the IDS to monitor the host and to respond the report as an alert. It establishes the roles and responsibilities for analyzing and monitoring the outcomes of both manual and automatic responses. Smart intruders who realize that IDS has been deployed on a network attack that they force it to provide false report.

Maintenance

Maintenance includes installation of signatures and IDS upgrades. Sensor placement should be revisited periodically to ensure that system or network changes. An organization must attract, train and retain qualified technical staff to operate and maintain IDS technologies.

1.9 Intrusion Detection In IDS

An IDS is based on the assumption that the intruder behaves differently from that of any legitimate users and these activities can be quantified. IDS reads network traffic and looks for patterns of attacks or signatures. If a signature is identified, IDS sends an alert to the Management Console and a response is immediately deployed. The following are the various approaches to intrusion detection:

- Statistical anomaly threshold detection

The approach where data is collected relating to the behavior of an authorized user. The data and the threshold are defined along with the frequency of occurrence for various events. A profile of activity is created of each user and is then developed for further use

to detect any significant behavior that would lead to a suspicious act. Any deviation from these legitimate or authorized profiles may send an alarm to the administrator.

- Rule based anomaly detection

This detection method defines a set of rules that can be used to define the type of behaviour of an intruder. These rules may represent past behavior patterns (of users, devices, timelines, etc.) which are matched to the current patterns to determine any intrusion.

This detection method is an expert system that defines a class of rules to identify suspicious behavior even when the behavior is confined to the patterns already established. These rules are more specific to the devices or the system.

The prime difference between the two is that the statistical approach attempts to define the normal or expected behavior and rule-based approach focuses on way the way the user navigates its way around the system.

1.10 IDS Challenges Today

Most of today's IDS products are focused on Signature Detection and are designed for sub-100 Mbps shared media network environments, employing detection capabilities introduced three to four years ago. IDS products have failed to keep up with the rapid advancement in switching and bandwidth growth and the increased sophistication of attacks as well as their sheer volume. Current IDS products often operate in a monitoring-only mode, sniffers, which can detect attacks but cannot effectively and reliably block malicious traffic before the damage is done.

Network security managers deploying IDS products today face a number of challenges:

- **Incomplete attack coverage:** IDS products typically focus on Signature or Anomaly or Denial of Service detection. Network security managers have to purchase and integrate point solutions from separate vendors, or leave networks vulnerable to attack.
- **Inaccurate detection:** IDS products detection capabilities can be characterized in terms of accuracy and specificity. Accuracy is often measured in true detection rate sometimes referred to as the false negative rate and the false positive rate. The true detection rate specifies how successful a system is in detecting attacks when they happen. The false positive rate tells us the likelihood that a system will misidentify benign activity as attacks. Specificity is a measure of how much detailed information about an attack is discovered when it is detected. IDS products today are lacking in both accuracy and specificity and generate too many

false positives, alerting security engineers of attacks, when nothing malicious is taking place. In some cases, IDS products have delivered tens of thousands of false positive alerts a day. There is nothing more corrosive to network vigilance than a jumpy security system, which is continually issuing false alarms.

- **Detection, not prevention:** Systems concentrate on attack detection. Preventing attacks is a reactive activity, often too late to thwart the intrusion.
- **Designed primarily for sub-100 Mbps networks:** Solutions have simply not kept up with the speed and sophistication network infrastructure; and cannot accurately monitor higher-speed or switched networks.
- **Performance challenged:** Software applications running on general purpose PC/server hardware do not have the processing power required to perform thorough analysis. These underpowered products result in inaccurate detection and packet dropping, even on low bandwidth networks.
- **Lack of high-availability deployment:** Single port products are not able to monitor asymmetric traffic flows. Also, with networks becoming a primary mechanism to interact with customers and partners, forward-thinking organizations have developed back-up systems should their current infrastructure fail in any way. The inability of current IDS products to cope with server failovers renders them virtually useless for any mission-critical network deployment.
- **Poor scalability:** Primarily designed for low-end deployments, today's IDS products do not scale for medium and large enterprise or government networks. Here monitored bandwidth, the number of network segments monitored, the number of sensors needed, alarm rates, and the geographical spread of the network exceeds system limits.
- **No multiple policy enforcement:** Current products generally support the selection of only one security policy for the entire system, even though the product may monitor traffic belonging to multiple administrative domains in an enterprise this could be the finance, marketing, or HR functions. This one size fits all approach is no longer acceptable for organizations that require different security policies for each function, business unit or geography.

- **Require significant IT resources:** IDS products today require substantial hands-on management for example, the simple task of frequent signature updates can take up a lot of time and skilled engineering resources, delivering a very high total cost of ownership.

In response to these limitations, a new architecture that detects and prevents known, unknown, and Denial of Service attacks was developed for home and small company networks.

2.1 Motivation For Intrusion Detection System

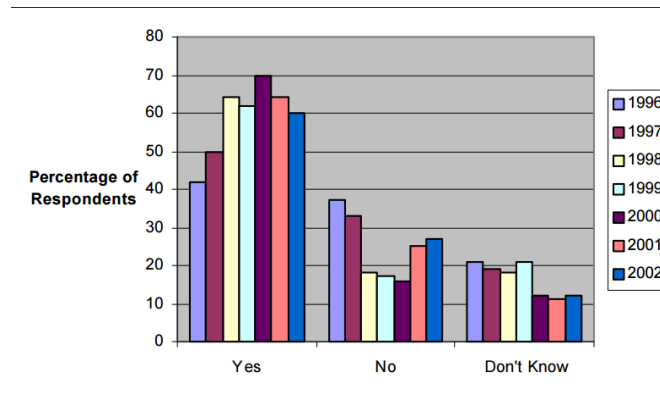


Fig 2.1: Unauthorized use of Computer Systems(source CSI/FBI Study)

In the year 2002, most common attacks were:

1. Virus (78%)
2. Insider Abuse of Net Access (78%)
3. Laptop Theft (55%)
4. Denial of Service and System Penetration (40%)
5. Unauthorised Access by Insiders (38%)

Apart from Laptop Theft which compromises about 55% of the attacks, all other attacks can be controlled or even decreased by using an Intrusion Detection System. These statistics are from a study conducted by CSI and FBI and they mark Virus attack and unauthorised net access as the top attack in these years.

2.2 Who are attacked?

Internet Information Services (IIS) web servers – which host web pages and serve them to users – are highly popular among business organizations, with over 6 million such servers installed worldwide. Unfortunately, IIS web servers are also popular among hackers and malicious fame-seekers – as a prime target for attacks!

As a result, every so often, new exploits emerge which endanger your IIS web server's integrity and stability. Many administrators have a hard time keeping up with the various security patches released for IIS to cope with each new exploit, making it easy for malicious users to find a vulnerable web server on the Internet. There are multiple issues

which can completely endanger your Web server – and possibly your entire corporate network and reputation.

People feel there is nothing on their system that anybody would want. But what they are unaware of is that, there is the issue of legal liability. You are potentially liable for damages caused by a hacker using your machine. You must be able to prove to a court that you took "reasonable" measures to defend yourself from hackers. For example, consider if you put a machine on a fast link (cable modem or DSL) and left administrator/root accounts open with no password. Then if a hacker breaks into that machine, then uses that machine to break into a bank, you may be held liable because you did not take the most obvious measures in securing the machine.

2.3 How they are attacked?

An intruder normally hacks into your system only after he has carefully accessed you and your security and he attacks you in a systematic way to cause maximum damage. The normal steps towards intrusion are:

- **Outside reconnaissance:** The intruder will find out as much as possible without actually giving himself away. They will do this by finding public information or appearing as a normal user. In this stage, you really can't detect them. The intruder will do a 'whois' lookup to find as much information as possible about your network as registered along with your Domain Name (such as foobar.com). The intruder might walk through your DNS tables (using 'nslookup', 'dig', or other utilities to do domain transfers) to find the names of your machines. The intruder will browse other public information, such as your public web sites and anonymous FTP sites. The intruder might search news articles and press releases about your company.
- **Inside reconnaissance:** The intruder uses more invasive techniques to scan for information, but still doesn't do anything harmful. They might walk through all your web pages and look for CGI scripts (CGI scripts are often easily hacked). They might do a 'ping' sweep in order to see which machines are alive. They might do a UDP/TCP scan/strobe on target machines in order to see what services are available. They'll run utilities like 'rcpinfo', 'showmount', 'snmpwalk', etc. in order to see what's available. At this point, the intruder has done 'normal' activity on the network and has not done anything that can be classified as an intrusion. At this point, a NIDS will be able to tell you that "somebody is checking door handles", but nobody has actually tried to open a door yet.
- **Exploit:** The intruder crosses the line and starts exploiting possible holes in the target machines. The intruder may attempt to compromise a CGI script by sending shell commands in input fields. The intruder might attempt to exploit well-known buffer-overrun holes by sending large amounts of data. The intruder may start checking for login accounts with easily guessable (or empty) passwords. The hacker may go through several stages of exploits. For example, if the hacker was

able to access a user account, they will now attempt further exploits in order to get root/admin access.

- **Foot hold:** At this stage, the hacker has successfully gained a foot hold in your network by hacking into a machine. The intruder's main goal is to hide evidence of the attacks (doctoring the audit trail and log files) and make sure they can get back in again. They may install 'toolkits' that give them access, replace existing services with their own Trojan horses that have backdoor passwords, or create their own user accounts. System Integrity Verifiers (SIVs) can often detect an intruder at this point by noting the changed system files. The hacker will then use the system as a stepping stone to other systems, since most networks have fewer defenses from inside attacks.
- **Profit:** The intruder takes advantage of their status to steal confidential data, misuse system resources (i.e. stage attacks at other sites from your site), or deface web pages.

3.1 Architecture



Fig 3.1: System Architecture

The system employs a small sized computer called Raspberry Pi. This device can be plugged into a computer monitor or TV. A mainstay in the world of makers and electronics, Raspberry Pi is a single-board, high-performance computer and can be simply turned into a powerful Honeypot or attack detector. Raspbian device comes with two models- Model A and Model B both priced at \$25 & \$35 respectively. Either of them could be used.

On top of this device we use the Raspbian operating System. The choice of Operating Sytem is important as we donot want out Raspberry to crash. So we only need an Operating System which offers minimal functionality so that we have only the main features of that OS. Other Operating System offering such minimal features are Arch Linux, Minibian etc. This operating System provides an interface between the user(administrator) and the hardware device.

Now to capture and analyze the data packets, we need a software or a tool. This functionality is provided by Snort. Snort is an open source tool created by Martin Roech used for intrusion detection. When suspicious behavior is detected, Snort sends a real-time alert to syslog, a separate 'alerts' file, or to a pop-up window.

3.2 Software Requirements:

- **Linux (Raspbian)**

Raspbian is a Debian-based computer operating system for Raspberry Pi. It is now officially provided by the Raspberry Pi Foundation, as the primary operating system for the family of Raspberry Pi single-board computers. Raspbian was created by Mike Thompson and Peter Green as an independent project. The initial build was completed in

June 2012. The operating system is still under active development. Raspbian is highly optimized for the Raspberry Pi line's low-performance ARM CPUs.

Raspbian uses PIXEL, **Pi Improved Xwindows Environment, Lightweight** as its main desktop environment as of the latest update. It is composed of a modified LXDE desktop environment and the Openbox stacking window manager with a new theme and few other changes. The distribution is shipped with a copy of computer algebra program Mathematica and a version of Minecraft called Minecraft Pi as well as a lightweight version of Chromium as of the latest version.

The latest version of Raspbian is Jessie which is based on Debian Jessie. This version was released in March 2, 2017.

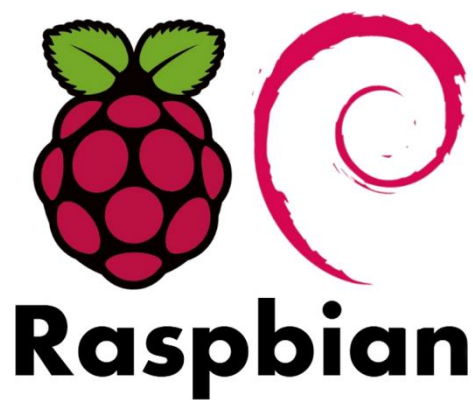


Fig 3.2: Raspbian Logo

- **Snort**

Snort is an open source tool used for intrusion detection. It was created by Martin Roech. It can be used as a network intrusion detection system that captures and scans network traffic in real time, examining each packet closely to detect an intrusion. Snort is based on libpcap (for library packet capture) a tool used in TCP/IP traffic sniffers and analyzers. Through protocol analysis and content searching and matching, Snort detects attack methods, including denial of service, buffer overflow, CGI attacks, stealthport scans, and SMB probes. When suspicious behavior is detected, Snort sends a real-time alert to syslog, a separate 'alerts' file, or to a pop-up window.



Fig 3.3: Snort Logo

Snort can be run in three different modes:

1. Packet Capture Mode- In packet capture mode Snort just looks into the packet header to determine if the packet is legitimate or not.
2. Logging Mode- In this mode of working, Snort logs the packet for later use.
3. Intrusion Detection Mode- In Intrusion Detection Mode, the Snort compares packets against rule base and decides if the packet is Legitimate or not.

Components of Snort may include:

- **Packet Decoder:** It processes packets from different interfaces for processing. Eg: Ethernet, PPP, SLIP etc. Packet Decoder also breaks down the packets so that it is understandable by Snort.
- **Preprocessors:** It modifies packets before detection engine. This is important as intruders can modify packet to evade detection. In short, Preprocessors can:
 - ✓ Decode HTTP URLs
 - ✓ Reassemble TCP streams
 - ✓ Defragment packets and reassemble
- **Detection Engine:** Detection Engine is used to detect intrusion activity in packets. It is time critical; may drop packets it can't handle. It applies predefined rules to TCP packets to detect malicious activity.
 - ✓ Rules may include IP header information (IP address).
 - ✓ TCP level header information (TCP & UDP).
 - ✓ Application layer header information (DNS, SSH, FTP, etc.)
 - ✓ Packet contents.
- **Logging and Alerting:** In this module, alerts are logged. Logged alerts may include header, content etc.
- **Output Modules:** This defines the control format of Logs
 - ✓ Text
 - ✓ Binary (to be read by tcp dump).
 - ✓ Log to db.
 - ✓ Generate XML.

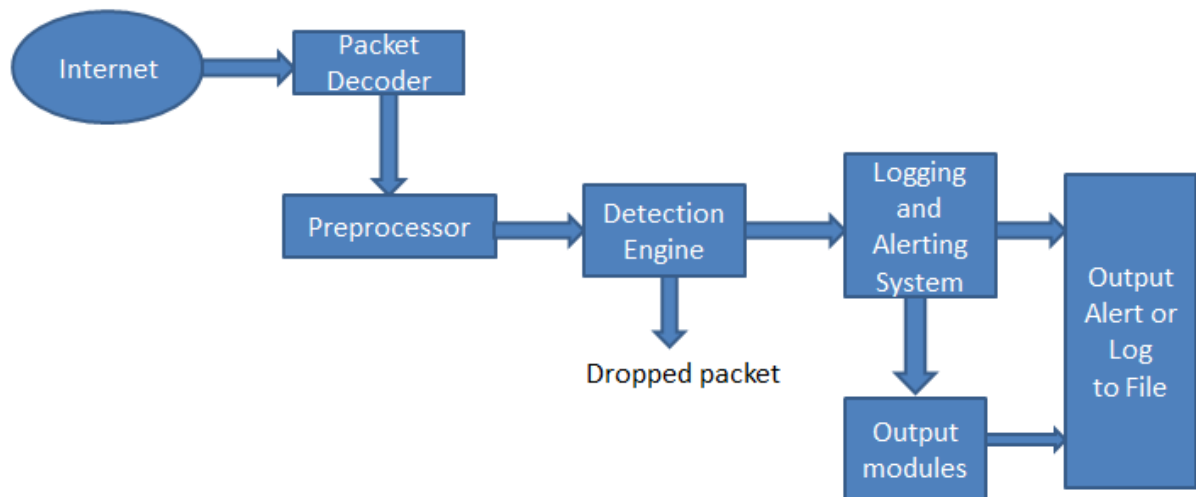


Fig 3.4: Components of Snort

- **PuTTY**

PuTTY is a free and open-source terminal emulator, serial console and network file transfer application. It supports several network protocols, including SCP, SSH, Telnet, rlogin, and raw socket connection. It can also connect to a serial port. The name "PuTTY" has no definitive meaning.

PuTTY was originally written for Microsoft Windows, but it has been ported to various other operating systems. Official ports are available for some Unix-like platforms, with work-in-progress ports to Classic Mac OS and macOS, and unofficial ports have been contributed to platforms such as Symbian, Windows Mobile and Windows Phone.

- **ZENMap**

Nmap (Network Mapper) is a security scanner, originally written by Gordon Lyon (also known by his pseudonym Fyodor Vaskovich), used to discover hosts and services on a computer network, thus building a "map" of the network. To accomplish its goal, Nmap sends specially crafted packets to the target host(s) and then analyzes the responses.

The software provides a number of features for probing computer networks, including host discovery and service and operating-system detection. These features are extensible by scripts that provide more advanced service detection, vulnerability detection, and other features. Nmap can adapt to network conditions including latency and congestion during a scan. The Nmap user community continues to develop and refine the tool.

Nmap started as a Linux-only utility, but porting to Windows, Solaris, HP-UX, BSD variants (including OS X), AmigaOS, and IRIX have followed. Linux is the most popular platform, followed closely by Windows.

- **Django**

Django is a web development framework that assists in building and maintaining quality web applications. Django helps eliminate repetitive tasks making the development process an easy and time saving experience. It is a high-level Python web framework that encourages rapid development and clean, pragmatic design. Django makes it easier to build better web apps quickly and with less code. As Django is a Python web framework. And like most modern framework, Django supports the MVC pattern. MVC is Model View Controller. It is a software design pattern for developing web applications. A Model View Controller pattern is made up of the following three parts:

- ✗ Model - The lowest level of the pattern which is responsible for maintaining data.
- ✗ View - This is responsible for displaying all or a portion of the data to the user.
- ✗ Controller - Software Code that controls the interactions between the Model and View.



Fig 3.5: Django Framework

- **Iperf**

Iperf is a tool used to measure the performance of a network. The developers NLNR and DAST created Iperf as a more modern alternative for measuring a networks functionality and throughput capacity. Iperf can be used to measure the maximum bandwidth with both TCP and UDP traffic. It does also report delay jitter and datagram loss. Iperf is available for most modern operating system such as Windows, MacOS X and Linux/Unix. Iperf is a client-server software where one side is a server and the other side is a client. The two parts connects and network traffic is sent between them to measure the desired data. The method can be configured in a number of ways which provides great flexibility for the users.

3.3 Hardware requirement:

- Raspberry Pi

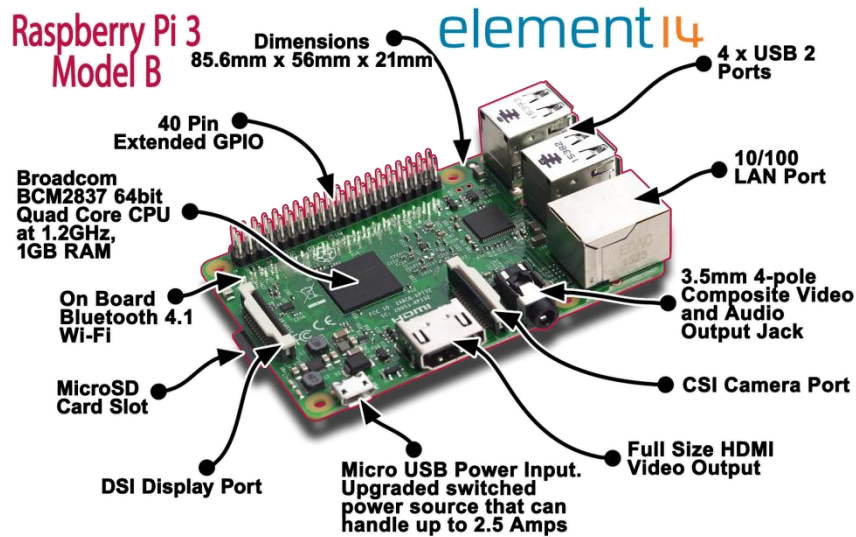


Fig 3.6 Raspberry Pi Model

The Raspberry Pi is a series of small single-board computers developed in the United Kingdom by the Raspberry Pi Foundation to promote the teaching of basic computer science in schools and in developing countries. The original model became far more popular than anticipated, selling outside of its target market for uses such as robotics. Peripherals (including keyboards, mice and cases) are not included with the Raspberry Pi. Some accessories however have been included in several official and unofficial bundles.

According to the Raspberry Pi Foundation, over 5 million Raspberry Pis have been sold before February 2015, making it the best-selling British computer. By 9 September 2016 they had sold 10 million

The Raspberry Pi 3 is the third generation Raspberry Pi. It replaced the Raspberry Pi 2 Model B in February 2016. Compared to the Raspberry Pi 2 it has:

- 1.2GHz 64-bit quad-core ARMv8 CPU
- 802.11n Wireless LANs
- Bluetooth 4.1 (BLE)
- 1GB RAM
- 4 USB ports
- Ethernet port
- 40 GPIO pins
- Full HDMI port
- Display interface (DSI)

➤ VideoCore IV 3D graphics core

- **Modular Connector**

A modular connector is an electrical connector that was originally designed for use in telephone wiring, but has since been used for many other purposes. Many applications that originally used a bulkier, more expensive connector have converted to modular connectors. Probably the most well-known applications of modular connectors are for telephone jacks and for Ethernet jacks, both of which are nearly always modular connectors.

Modular connectors were originally used in the Registration Interface system, mandated by the Federal Communications Commission (FCC) in 1976 in which they became known as registered jacks. The registered jack specifications define the wiring patterns of the jacks, not the physical dimensions or geometry of the connectors of either gender. Instead, these latter aspects are covered by ISO standard 8877, first used in ISDN systems. TIA/EIA-568 is a standard for data circuits wired on modular connectors.

Other systems exist for assigning signals to modular connectors; physical interchangeability of plugs and jacks does not ensure interoperability, nor protection from electrical damage to circuits. For example, modular cables and connectors have been used to supply low-voltage AC or DC power and no clear standard exists for this application.

- **Display Device**

A display device is an output device for presentation of information in visual or tactile form (the latter used for example in tactile electronic displays for blind people). When the input information is supplied has an electrical signal, the display is called an electronic display.

3.4 Practicality of Hardware

In 2013 a study was made on the possibility of using a Raspberry as an IDS in a home network. They performance tested a Raspberry Pi model B+ running the operating system IPFire and the intrusion detection software Snort. The results from this study showed that a Raspberry Pi could be used as an IDS, but with the following problems:

- A limit on how many Snort-rules could be used due to limitations in memory.
- A noticeable degradation in throughput when Snort was active.

We want to examined the possibility of implementing an intrusion detection system, on a Raspberry Pi model 3, that does not suffer from such limitations, as operating system is

Raspbian Minimal which is lightweight and uses low system resources. We also want to added the functionalities of a router to this hardware using python and django to run the webservice.

4.1 Use Case Diagram

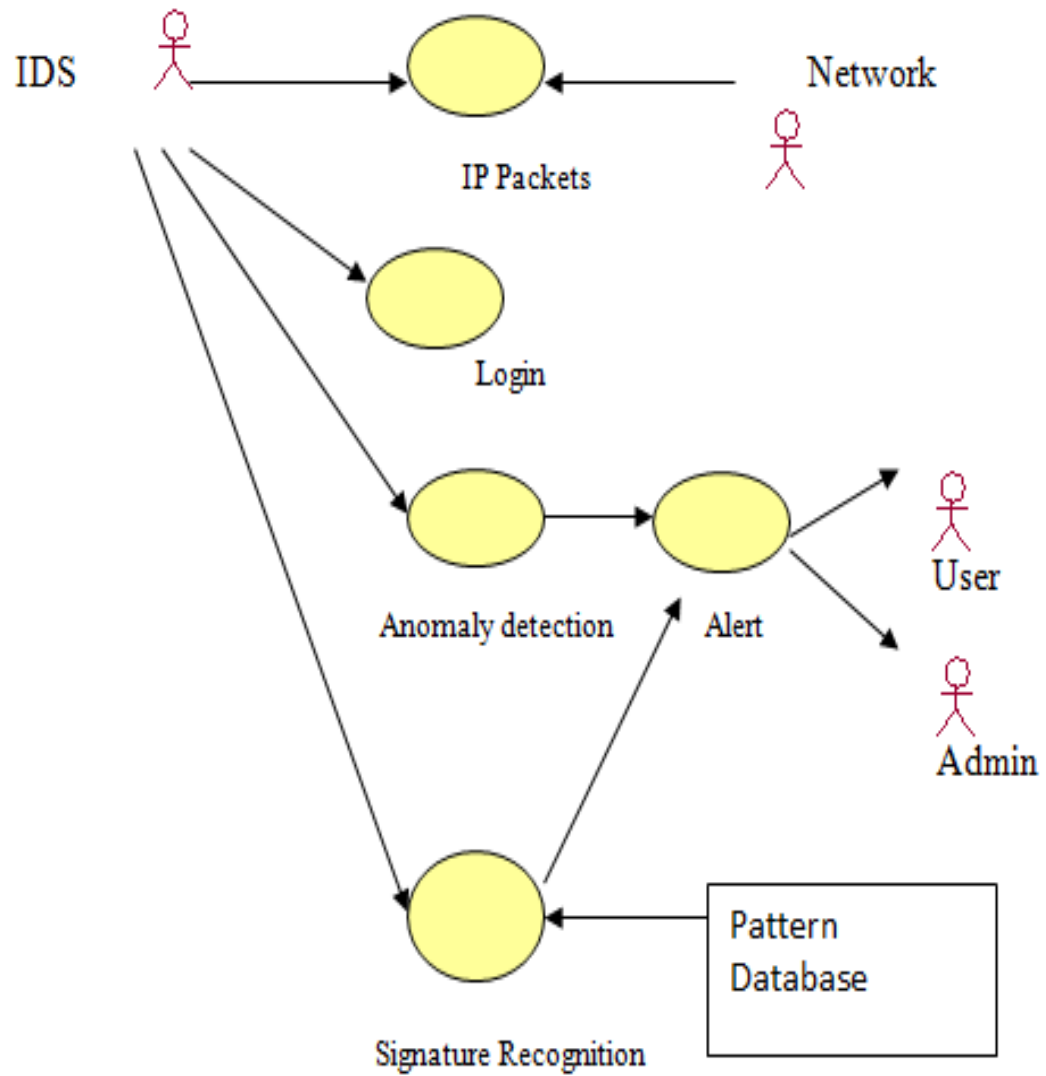


Fig 4.1: Use Case Diagram

Use Cases Actors:

- 1) User: User sends request to server and server responds by providing the requested service.
- 2) Network: Network carries the IP packets from source to destination.
- 3) IDS: IDS takes the packets from the network, analyses the packets.
- 4) System Administrator: System Administrator is alerted by the IDS of any suspicious activity or whenever intrusion is detected.

Use Case Description:

- 1) IP Packets: Network gives the IP Packets to IDS which does further processing of these packets.
- 2) Anomaly Detection: If IDS detects any abnormality in the network traffic, then it triggers the alert system
- 3) Signature recognition: IDS examines the traffic looking for well-known patterns of attack, which are saved in pattern database and triggers the alert system, if a match is found.
- 4) Alert System: Whenever triggered by anomaly detection or signature recognition, it alerts the system administrator.

4.2 Data Flow Diagram

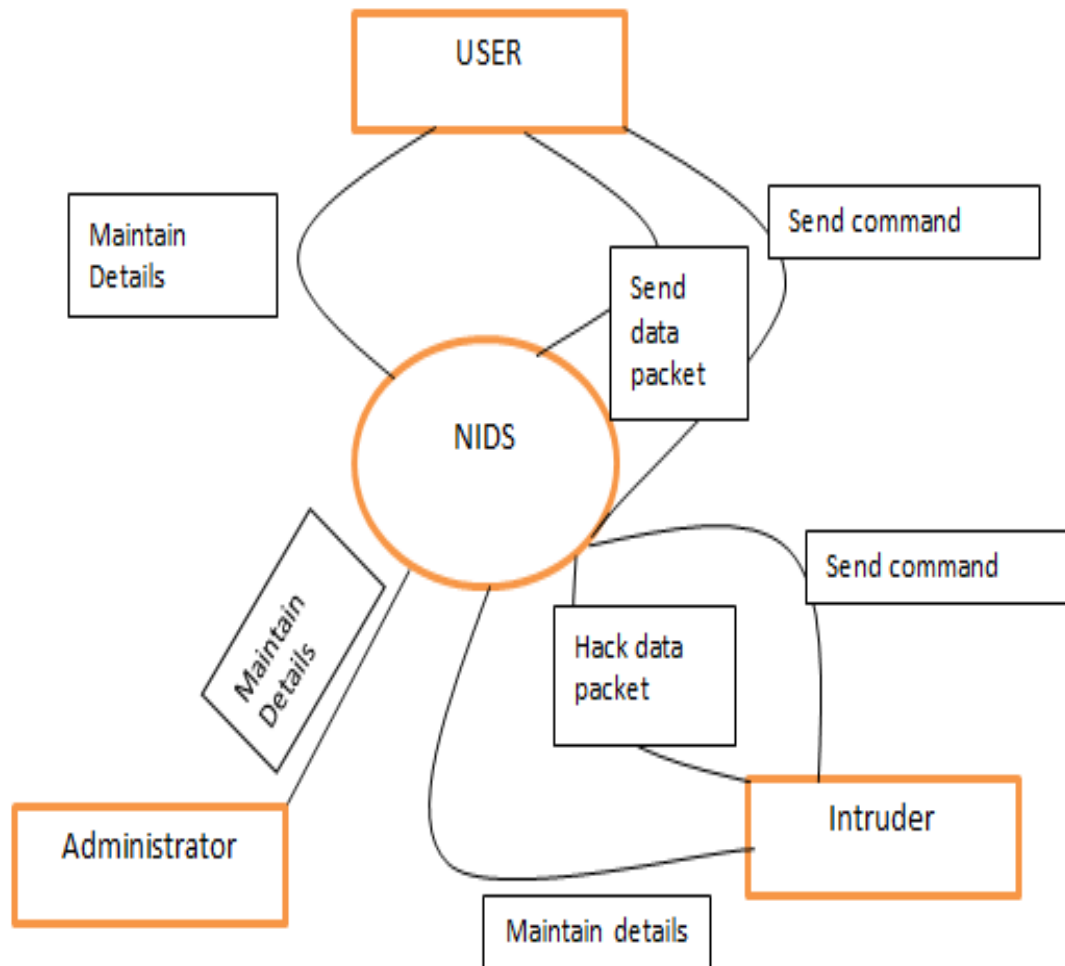


Fig 4.2: Data Flow Diagram

4.3 E-R Diagram For IDS

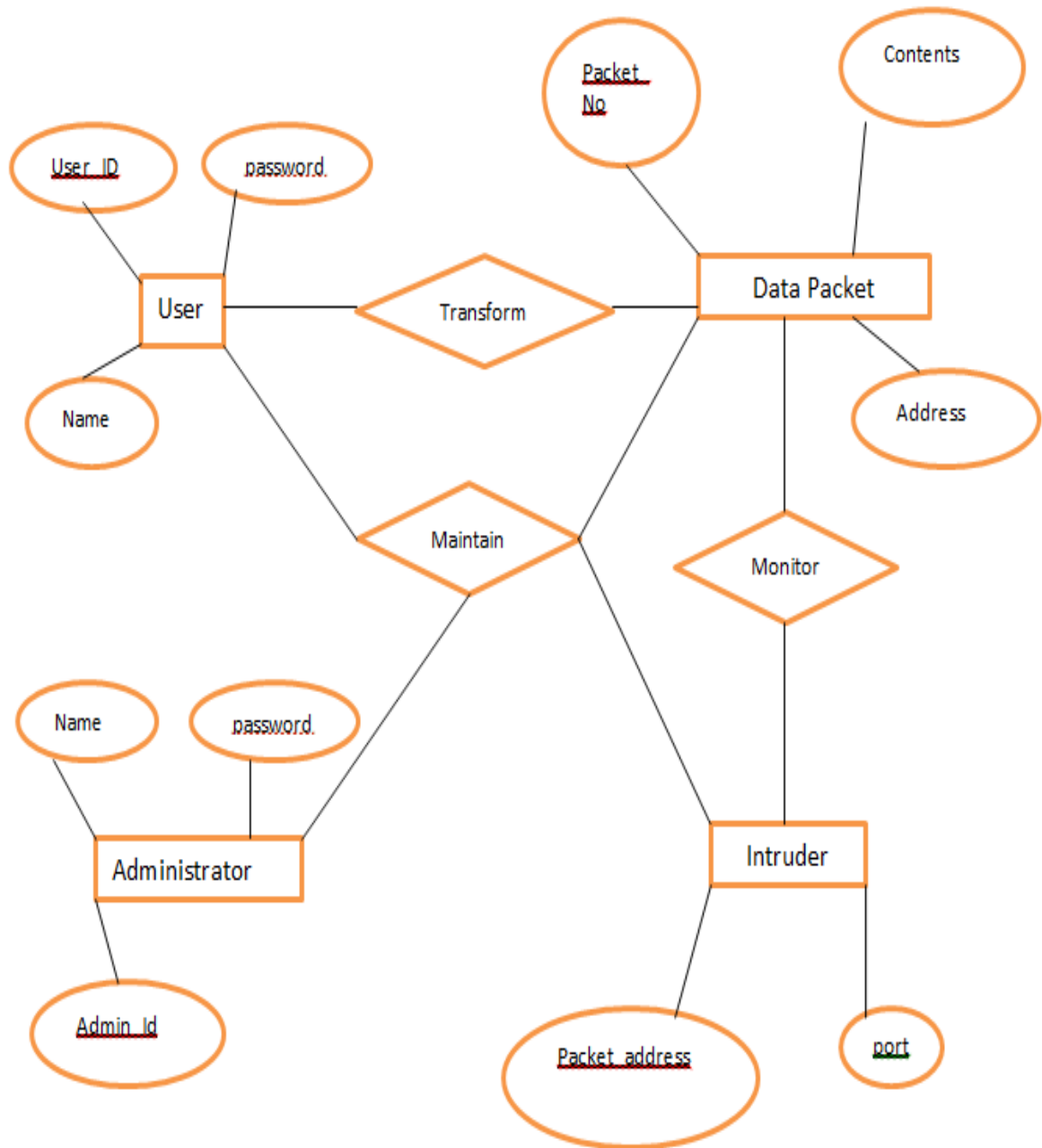


Fig 4.3: E-R Diagram

4.4 Implementation

Preparation

1. OS(minimal Raspbian) was installed onto your SD card
2. Pi was configured for ssh and other services
3. Ethernet and Wifi connection was tested.
4. SD card was expanded for storage

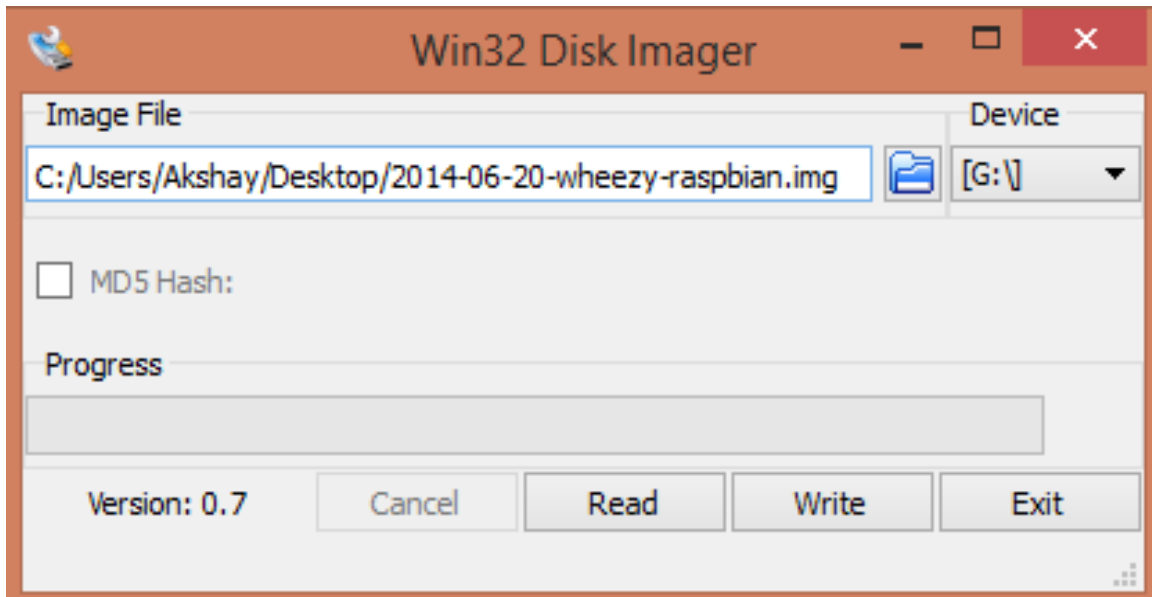


Fig 4.4: Installing Raspbian OS

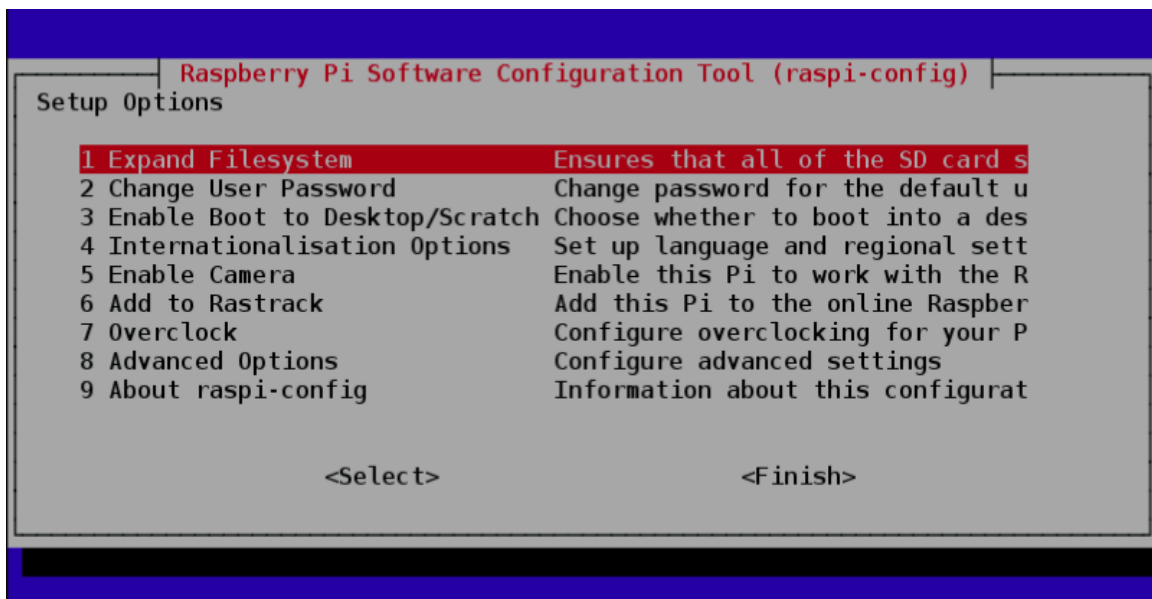


Fig 4.5: Setting up Pi

Install necessary Softwares

```
sudo apt-get update
```

```
sudo apt-get update
```

```
sudo apt-get install hostapd isc-dhcp-server
```

```
sudo apt-get install iptables-persistent
```

```
sudo apt-get install snort
```

DHCP server Configuration

```
#option domain-name "example.org";
```

```
#option domain-name-servers ns1.example.org, ns2.example.org;
```

```
authoritative;
```

```
subnet 192.168.42.0 netmask 255.255.255.0 {
```

```
    range 192.168.42.10 192.168.42.50;
```

```
    option broadcast-address 192.168.42.255;
```

```
    option routers 192.168.42.1;
```

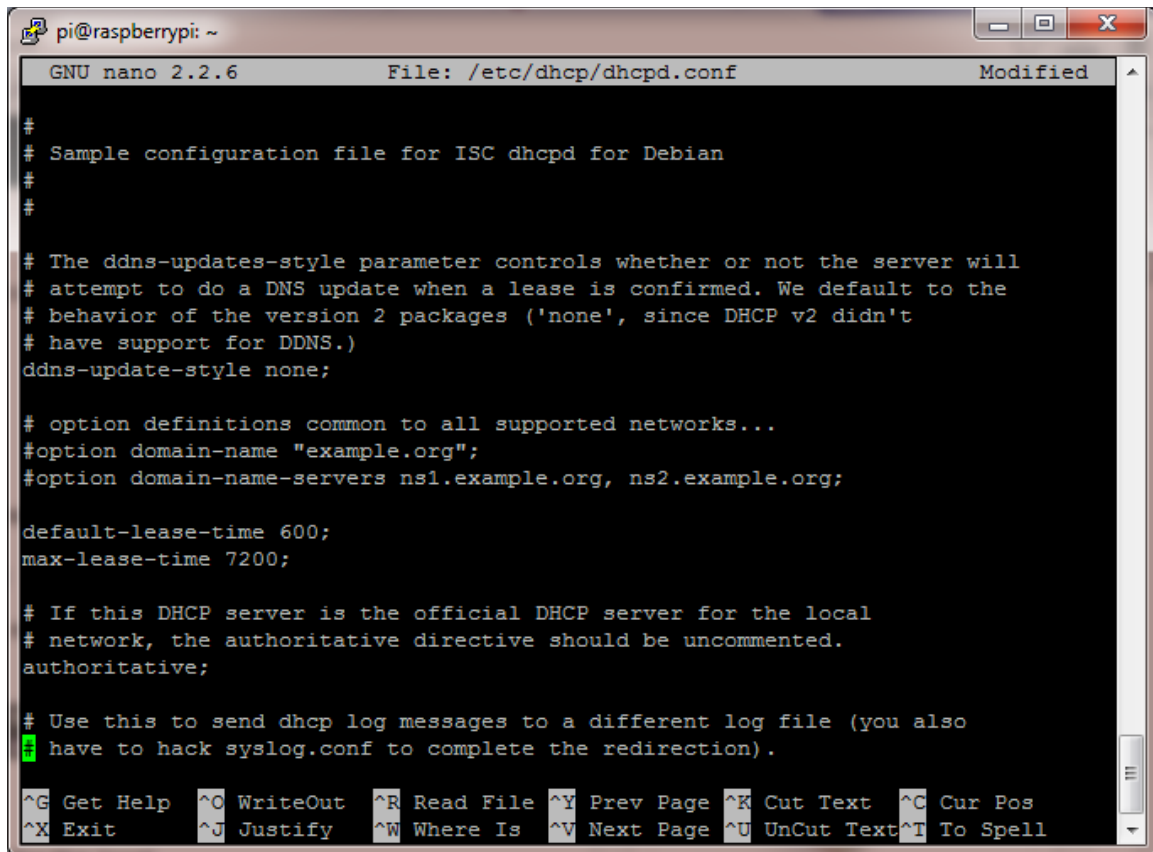
```
    default-lease-time 600;
```

```
    max-lease-time 7200;
```

```
    option domain-name "local";
```

```
    option domain-name-servers 8.8.8.8, 8.8.4.4;
```

```
}
```



The screenshot shows a terminal window on a Raspberry Pi with the title bar 'pi@raspberrypi: ~'. The window contains the GNU nano 2.2.6 editor editing the file /etc/dhcp/dhcpd.conf. The file is a sample configuration for ISC dhcpd for Debian. The configuration includes comments about ddns-update-style, option definitions for domain-name and domain-name-servers, default-lease-time, max-lease-time, authoritative, and logging. The bottom of the window shows a status bar with various keyboard shortcuts like ^G Get Help, ^O WriteOut, ^R Read File, ^Y Prev Page, ^K Cut Text, ^C Cur Pos, ^X Exit, ^J Justify, ^W Where Is, ^V Next Page, ^U UnCut Text, and ^T To Spell.

```
pi@raspberrypi: ~
GNU nano 2.2.6      File: /etc/dhcp/dhcpd.conf      Modified
#
# Sample configuration file for ISC dhcpd for Debian
#
#
# The ddns-update-style parameter controls whether or not the server will
# attempt to do a DNS update when a lease is confirmed. We default to the
# behavior of the version 2 packages ('none', since DHCP v2 didn't
# have support for DDNS.)
ddns-update-style none;

# option definitions common to all supported networks...
#option domain-name "example.org";
#option domain-name-servers ns1.example.org, ns2.example.org;

default-lease-time 600;
max-lease-time 7200;

# If this DHCP server is the official DHCP server for the local
# network, the authoritative directive should be uncommented.
authoritative;

# Use this to send dhcp log messages to a different log file (you also
# have to hack syslog.conf to complete the redirection).

^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text ^T To Spell
```

Fig 4.6: Installing & Configuring DHCP Server

Ethernet and Wireless Interface configuration

auto lo

iface lo inet loopback

iface eth0 inet dhcp

allow-hotplug wlan0

iface wlan0 inet static

address 192.168.42.1

netmask 255.255.255.0

network 192.168.42.1

Access Point Configuration

interface=wlan0

hw_mode=g

macaddr_acl=0

auth_algs=1

wpa=2

wpa_key_mgmt=WPA-PSK

wpa_pairwise=CCMP

wpa_group_rekey=86400

ieee80211n=1

wme_enabled=1

ssid=PIDS

channel=2

wpa_passphrase=1234567890

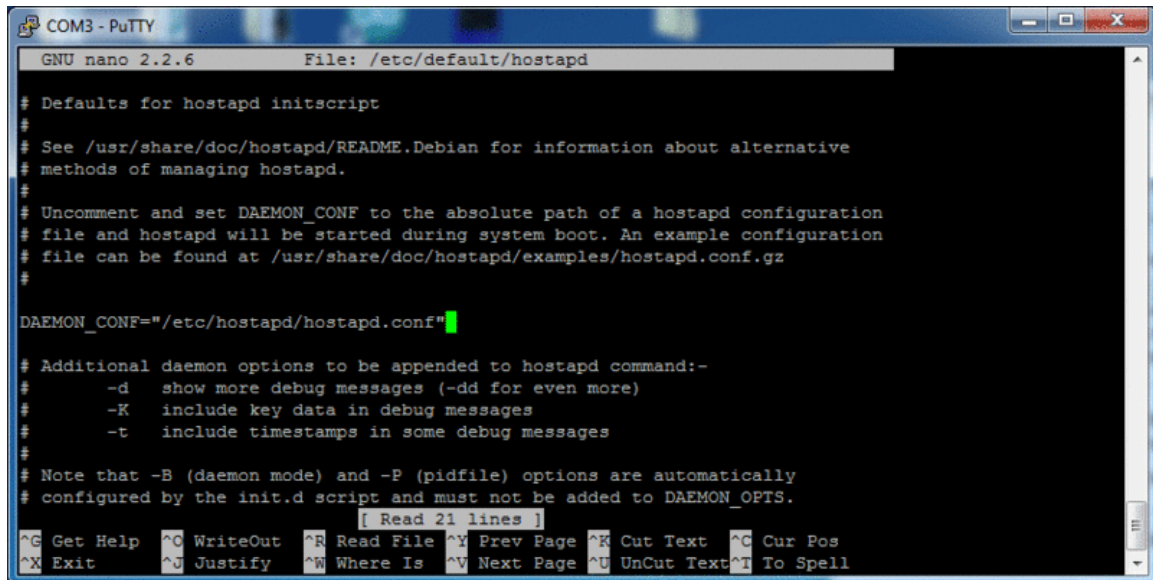
ignore_broadcast_ssid=0

Network Address Transltion

sudo iptables -t nat -A POSTROUTING -o eth0 -j MASQUERADE

sudo iptables -A FORWARD -i eth0 -o wlan0 -m state --state
RELATED,ESTABLISHED -j ACCEPT

sudo iptables -A FORWARD -i wlan0 -o eth0 -j ACCEPT



```
COM3 - PuTTY
GNU nano 2.2.6      File: /etc/default/hostapd

# Defaults for hostapd initscript
#
# See /usr/share/doc/hostapd/README.Debian for information about alternative
# methods of managing hostapd.
#
# Uncomment and set DAEMON_CONF to the absolute path of a hostapd configuration
# file and hostapd will be started during system boot. An example configuration
# file can be found at /usr/share/doc/hostapd/examples/hostapd.conf.gz
#
DAEMON_CONF="/etc/hostapd/hostapd.conf"

# Additional daemon options to be appended to hostapd command:-
#
#   -d  show more debug messages (-dd for even more)
#   -K  include key data in debug messages
#   -t  include timestamps in some debug messages
#
# Note that -B (daemon mode) and -P (pidfile) options are automatically
# configured by the init.d script and must not be added to DAEMON_OPTS.
[ Read 21 lines ]
^G Get Help  ^O WriteOut  ^R Read File ^Y Prev Page ^K Cut Text  ^C Cur Pos
^X Exit      ^J Justify   ^W Where Is  ^V Next Page ^U UnCut Text ^T To Spell
```

Fig 4.7: Configuring HostAPD

Snort Installation and Setup

```
sudo apt-get install snort
```

```
ipvar HOME_NET 192.168.42.0/24
```

```
var RULE_PATH rules
```

```
include $RULE_PATH/ftp.rules
```

```
include $RULE_PATH/community-ftp.rules
```

```
include $RULE_PATH/bad-traffic.rules
```

```
include $RULE_PATH/telnet.rules
```

```
include $RULE_PATH/iplists/black_list.rules
```

```
include $RULE_PATH/exploit.rules
```

```
include $RULE_PATH/community-exploit.rules
```

```
include $RULE_PATH/scan.rules
```

```
include $RULE_PATH/community-icmp.rules
```

```
include $RULE_PATH/icmp.rules
```

```
include $RULE_PATH/multimedia.rules
```

```
include $RULE_PATH/virus.rules
```

```
include $RULE_PATH/sql.rules
```

```
include $RULE_PATH/community-sql-injection.rules
```

```
include $RULE_PATH/smtp.rules
```

```
include $RULE_PATH/community-smtp.rules
```

```
include $RULE_PATH/community-mail-client.rules
```

```
include $RULE_PATH/web-attacks.rules
```

```
include $RULE_PATH/community-web-attacks.rules
```

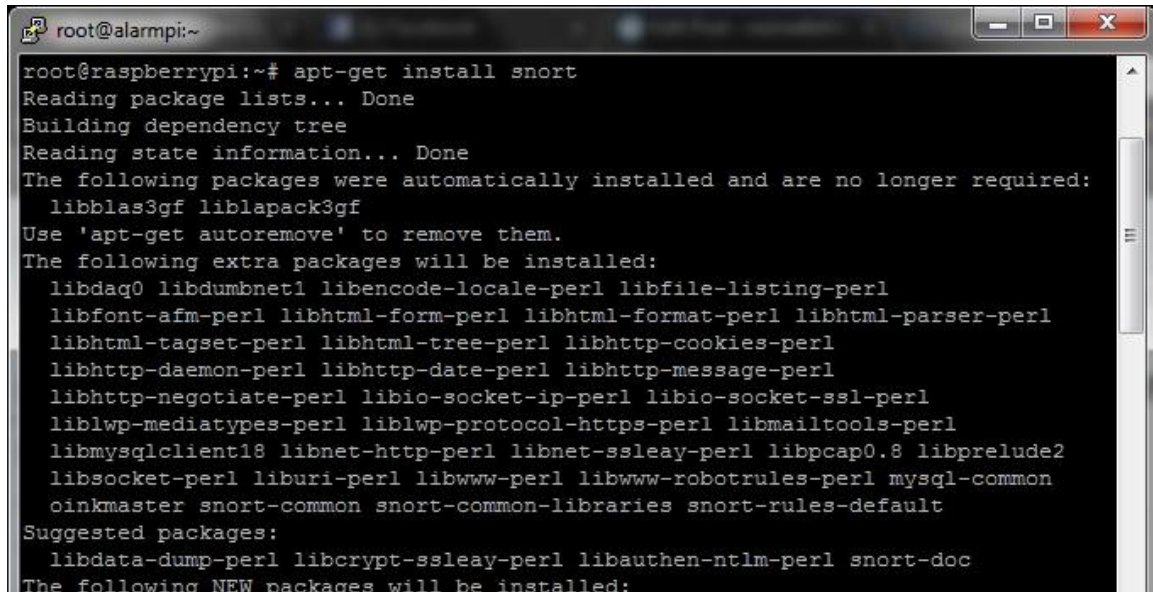
```
include $RULE_PATH/community-web-client.rules
```

```
include $RULE_PATH/custom.rules
```

Editing rc.local

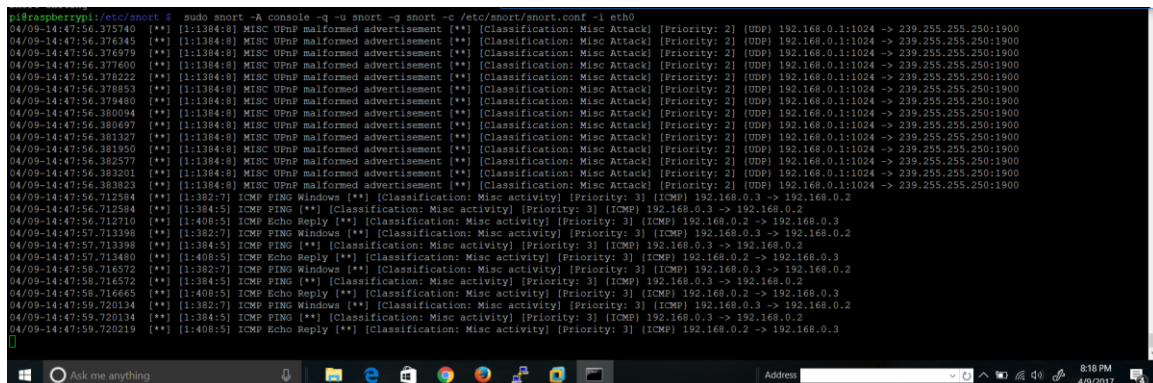
```
sudo python /home/pi/AdminPanel/manage.py runserver 192.168.42.1:8080
```

```
sudo snort -b -c /etc/snort/snort.conf -l /var/log/snort
```



```
root@alarmpi:~  
root@raspberrypi:~# apt-get install snort  
Reading package lists... Done  
Building dependency tree  
Reading state information... Done  
The following packages were automatically installed and are no longer required:  
  libblas3gf liblapack3gf  
Use 'apt-get autoremove' to remove them.  
The following extra packages will be installed:  
  libdaq0 libdumbnet1 libencode-locale-perl libfile-listing-perl  
  libfont-afm-perl libhtml-form-perl libhtml-format-perl libhtml-parser-perl  
  libhtml-tagset-perl libhtml-tree-perl libhttp-cookies-perl  
  libhttp-daemon-perl libhttp-date-perl libhttp-message-perl  
  libhttp-negotiate-perl libio-socket-ip-perl libio-socket-ssl-perl  
  liblwp-mediatypes-perl liblwp-protocol-https-perl libmailtools-perl  
  libmysqlclient18 libnet-http-perl libnet-ssleay-perl libpcap0.8 libprelude2  
  libsocket-perl liburi-perl libwww-perl libwww-robotrules-perl mysql-common  
  oinkmaster snort-common snort-common-libraries snort-rules-default  
Suggested packages:  
  libdata-dump-perl libcrypt-ssleay-perl libauthen-ntlm-perl snort-doc  
The following NEW packages will be installed:
```

Fig 4.8: Installing Snort



```
pi@raspberrypi:/etc/snort$ sudo snort -A console -q -u snort -g snort -c /etc/snort/snort.conf -i eth0  
04/09-14:47:56.375740 ** [1:1384:8] MISC UDP malformed advertisement ** [Classification: Misc Attack] (Priority: 2) (UDP) 192.168.0.1:1024 -> 239.255.255.250:1900  
04/09-14:47:56.376345 ** [1:1384:8] MISC UDP malformed advertisement ** [Classification: Misc Attack] (Priority: 2) (UDP) 192.168.0.1:1024 -> 239.255.255.250:1900  
04/09-14:47:56.376979 ** [1:1384:8] MISC UDP malformed advertisement ** [Classification: Misc Attack] (Priority: 2) (UDP) 192.168.0.1:1024 -> 239.255.255.250:1900  
04/09-14:47:56.377600 ** [1:1384:8] MISC UDP malformed advertisement ** [Classification: Misc Attack] (Priority: 2) (UDP) 192.168.0.1:1024 -> 239.255.255.250:1900  
04/09-14:47:56.378222 ** [1:1384:8] MISC UDP malformed advertisement ** [Classification: Misc Attack] (Priority: 2) (UDP) 192.168.0.1:1024 -> 239.255.255.250:1900  
04/09-14:47:56.378853 ** [1:1384:8] MISC UDP malformed advertisement ** [Classification: Misc Attack] (Priority: 2) (UDP) 192.168.0.1:1024 -> 239.255.255.250:1900  
04/09-14:47:56.379480 ** [1:1384:8] MISC UDP malformed advertisement ** [Classification: Misc Attack] (Priority: 2) (UDP) 192.168.0.1:1024 -> 239.255.255.250:1900  
04/09-14:47:56.380094 ** [1:1384:8] MISC UDP malformed advertisement ** [Classification: Misc Attack] (Priority: 2) (UDP) 192.168.0.1:1024 -> 239.255.255.250:1900  
04/09-14:47:56.380697 ** [1:1384:8] MISC UDP malformed advertisement ** [Classification: Misc Attack] (Priority: 2) (UDP) 192.168.0.1:1024 -> 239.255.255.250:1900  
04/09-14:47:56.381327 ** [1:1384:8] MISC UDP malformed advertisement ** [Classification: Misc Attack] (Priority: 2) (UDP) 192.168.0.1:1024 -> 239.255.255.250:1900  
04/09-14:47:56.381950 ** [1:1384:8] MISC UDP malformed advertisement ** [Classification: Misc Attack] (Priority: 2) (UDP) 192.168.0.1:1024 -> 239.255.255.250:1900  
04/09-14:47:56.382577 ** [1:1384:8] MISC UDP malformed advertisement ** [Classification: Misc Attack] (Priority: 2) (UDP) 192.168.0.1:1024 -> 239.255.255.250:1900  
04/09-14:47:56.383201 ** [1:1384:8] MISC UDP malformed advertisement ** [Classification: Misc Attack] (Priority: 2) (UDP) 192.168.0.1:1024 -> 239.255.255.250:1900  
04/09-14:47:56.383823 ** [1:1384:8] MISC UDP malformed advertisement ** [Classification: Misc Attack] (Priority: 2) (UDP) 192.168.0.1:1024 -> 239.255.255.250:1900  
04/09-14:47:56.712584 ** [1:382:7] ICMP PING Windows ** [Classification: Misc activity] (Priority: 3) (ICMP) 192.168.0.3 -> 192.168.0.2  
04/09-14:47:56.712584 ** [1:384:5] ICMP PING ** [Classification: Misc activity] (Priority: 3) (ICMP) 192.168.0.3 -> 192.168.0.2  
04/09-14:47:56.712710 ** [1:408:5] ICMP Echo Reply ** [Classification: Misc activity] (Priority: 3) (ICMP) 192.168.0.2 -> 192.168.0.3  
04/09-14:47:57.713398 ** [1:382:7] ICMP PING Windows ** [Classification: Misc activity] (Priority: 3) (ICMP) 192.168.0.3 -> 192.168.0.2  
04/09-14:47:57.713398 ** [1:384:5] ICMP PING ** [Classification: Misc activity] (Priority: 3) (ICMP) 192.168.0.3 -> 192.168.0.2  
04/09-14:47:57.713480 ** [1:408:5] ICMP Echo Reply ** [Classification: Misc activity] (Priority: 3) (ICMP) 192.168.0.2 -> 192.168.0.3  
04/09-14:47:58.716572 ** [1:382:7] ICMP PING Windows ** [Classification: Misc activity] (Priority: 3) (ICMP) 192.168.0.3 -> 192.168.0.2  
04/09-14:47:58.716572 ** [1:384:5] ICMP PING ** [Classification: Misc activity] (Priority: 3) (ICMP) 192.168.0.3 -> 192.168.0.2  
04/09-14:47:58.716665 ** [1:408:5] ICMP Echo Reply ** [Classification: Misc activity] (Priority: 3) (ICMP) 192.168.0.2 -> 192.168.0.3  
04/09-14:47:59.720134 ** [1:382:7] ICMP PING Windows ** [Classification: Misc activity] (Priority: 3) (ICMP) 192.168.0.3 -> 192.168.0.2  
04/09-14:47:59.720134 ** [1:384:5] ICMP PING ** [Classification: Misc activity] (Priority: 3) (ICMP) 192.168.0.3 -> 192.168.0.2  
04/09-14:47:59.720219 ** [1:408:5] ICMP Echo Reply ** [Classification: Misc activity] (Priority: 3) (ICMP) 192.168.0.2 -> 192.168.0.3
```

```
Sudo snort -q -u snort -g snort -c /etc/snort/snort.conf -i eth0
```

Fig 4.9: Starting Snort

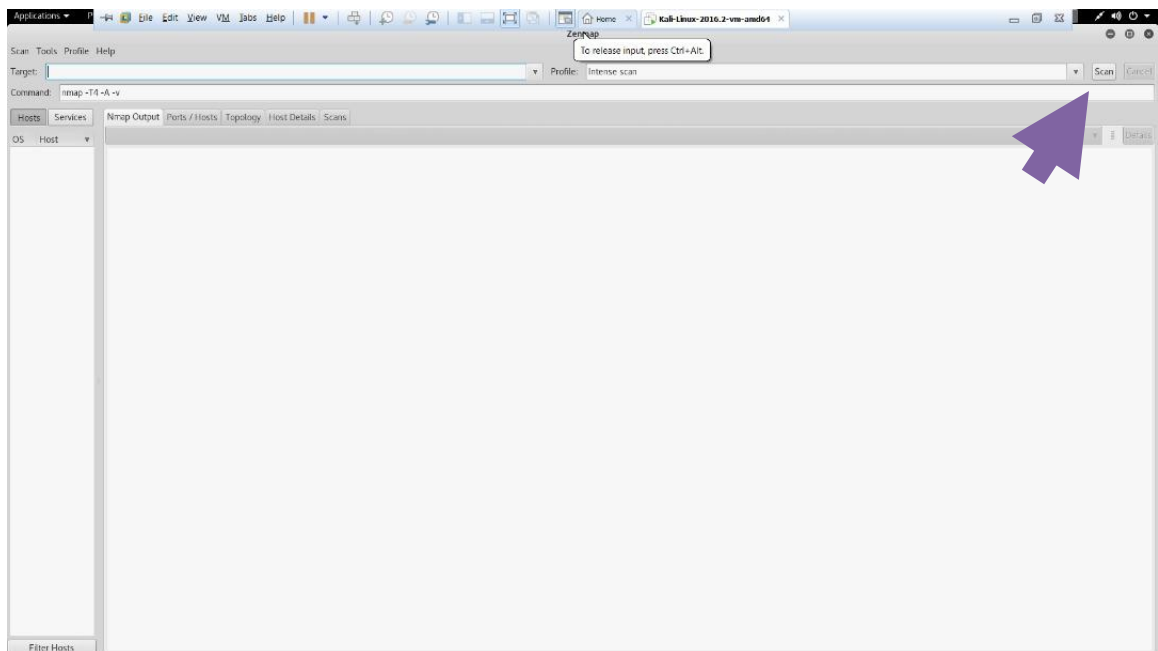
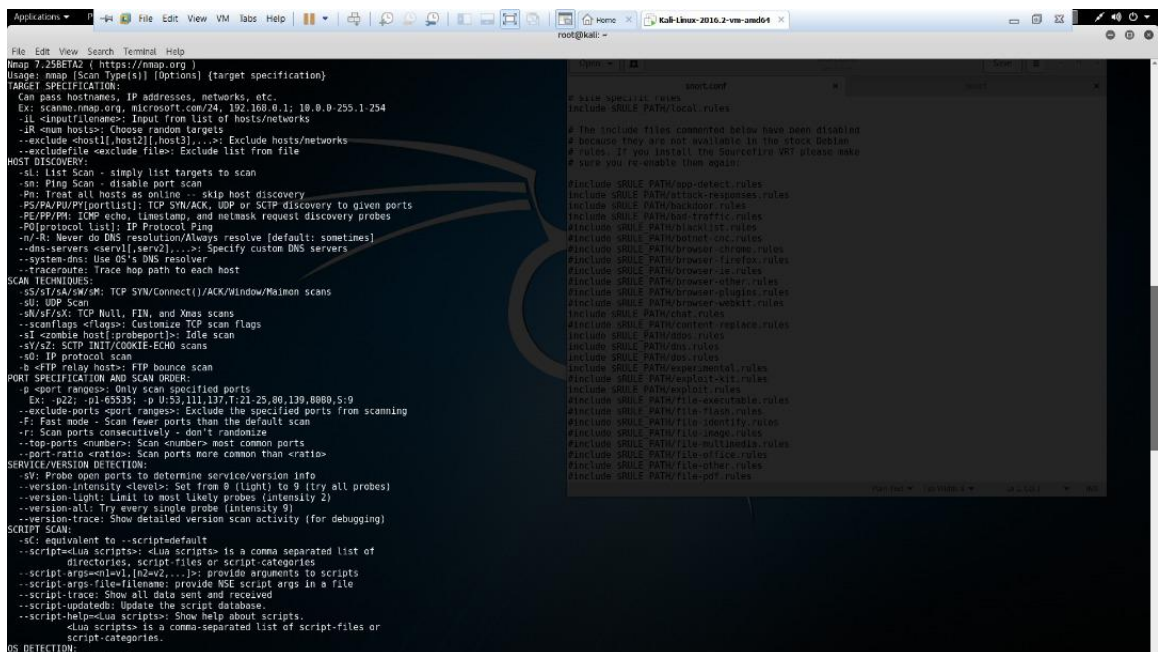


Fig 4.10: Nmap used to scan port to attempt an intrusion



```
nmap -sV -p 1-65535 192.168.1.1/24
```

Fig 4.11: nmap help contents

This command will scan all of your local IP range (assuming your in the 192.168.1.0-254 range), and will perform service identification (-sV) and will scan all ports (-p 1-65535). Since you are running this as a normal user and not root it will be TCP Connect based scan. If you run the command with `sudo` at the front it will run as a TCP SYN scan.

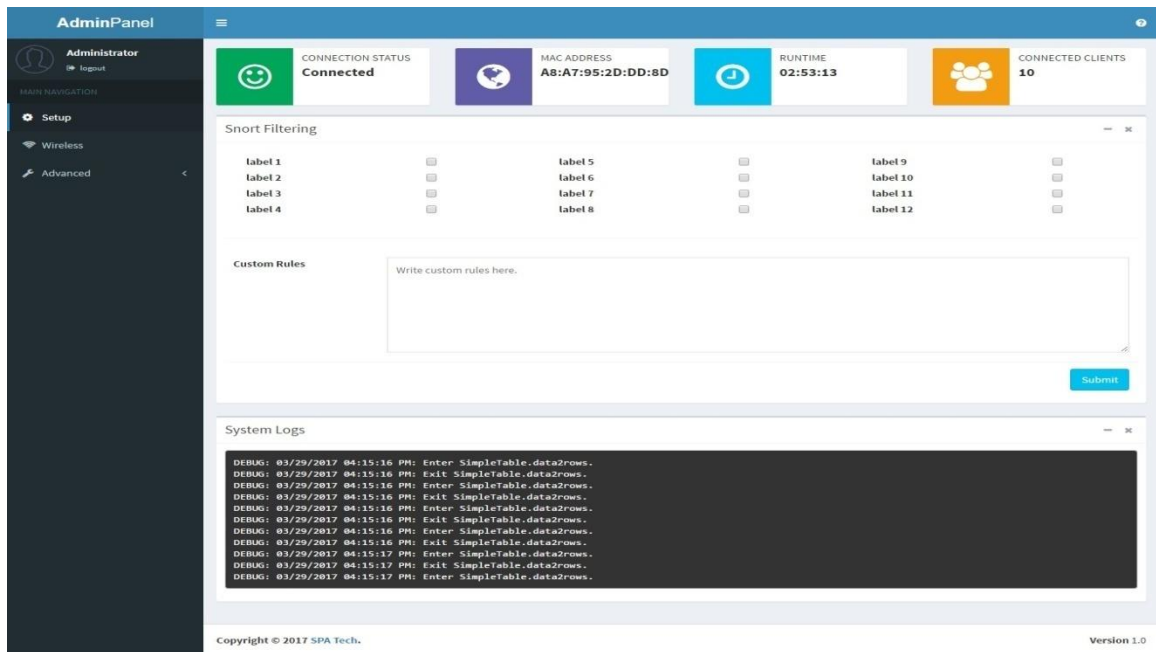


Fig 4.12: Manually Selecting Snort Rules

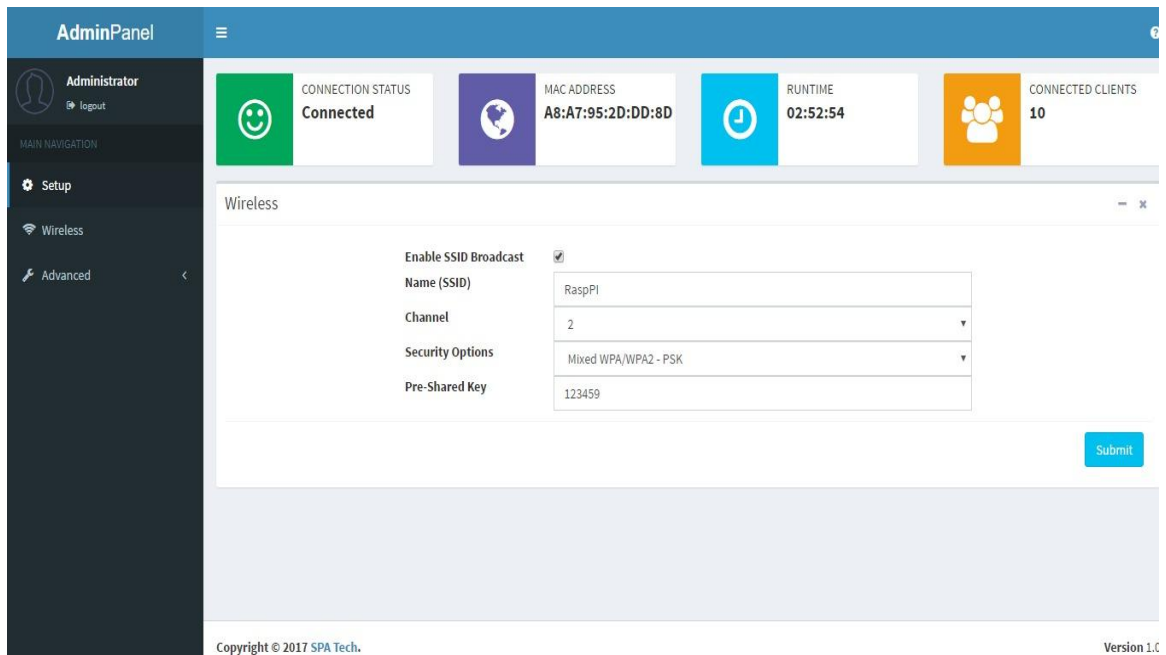


Fig 4.13: GUI based Admin Panel

5.1 Testing

Topology

For each test a Raspberry Pi was connected between the physical machines with an Ethernet and/or wireless connection. As the Raspberry Pis' only have one physical Ethernet port, and USB-To-Ethernet adapter was used to enable them to connect to one of the virtual machines through its USB-port. Raspberry Pis ran the operating system Raspbian Minimal and had the intrusion detection software Snort installed. The configuration for Snort was identical on both Raspberry Pi and both had IPv4-forwarding enabled in the kernel.

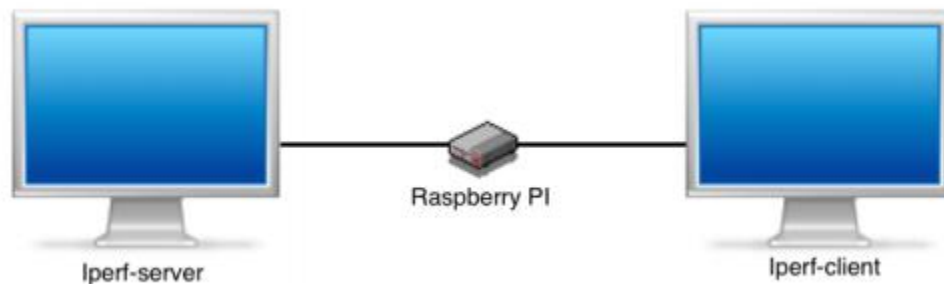


Fig 5.1 System Topology

Measurement

Iperf was the tool used to generate traffic to measure bandwidth capacity and throughput. We chose to use this tool because it gives us the ability to generate the traffic we need at will and provides it consistently. Iperf can be used to generate both TCP and UDP traffic between an Iperf server and an Iperf client.

On the server side the following command was used:

```
Iperf -s
```

The command runs Iperf in server mode with port 5001 open. This makes it possible for an Iperf client to connect to the Iperf server on port 5001 and sends TCP-packets to the server.

On the client side the following command was used:

```
Iperf -c 192.168.1.10 -t 300
```

The command runs Iperf in client mode and thereby connects to the Iperf server on the default port 5001. It then sends TCP-packets for 300 seconds. We chose to use the default setting for how much data to be transferred at a time. The default is to send 85KB sized TCP-packets a number of times. As we chose 300 seconds it will send 85KB sized TCP packets as many times as is possible for 300 seconds.

Security Testing

`nmap -sV -p 1-65535 192.168.42.1/24`

This command was used to scan all local IP range and performed service identification (-sV) and scanned all ports (-p 1-65535). Since it was running as a normal user and not root it scanned TCP Connect based scan. The command was then executed with sudo at the front it hence performed as a TCP SYN scan.

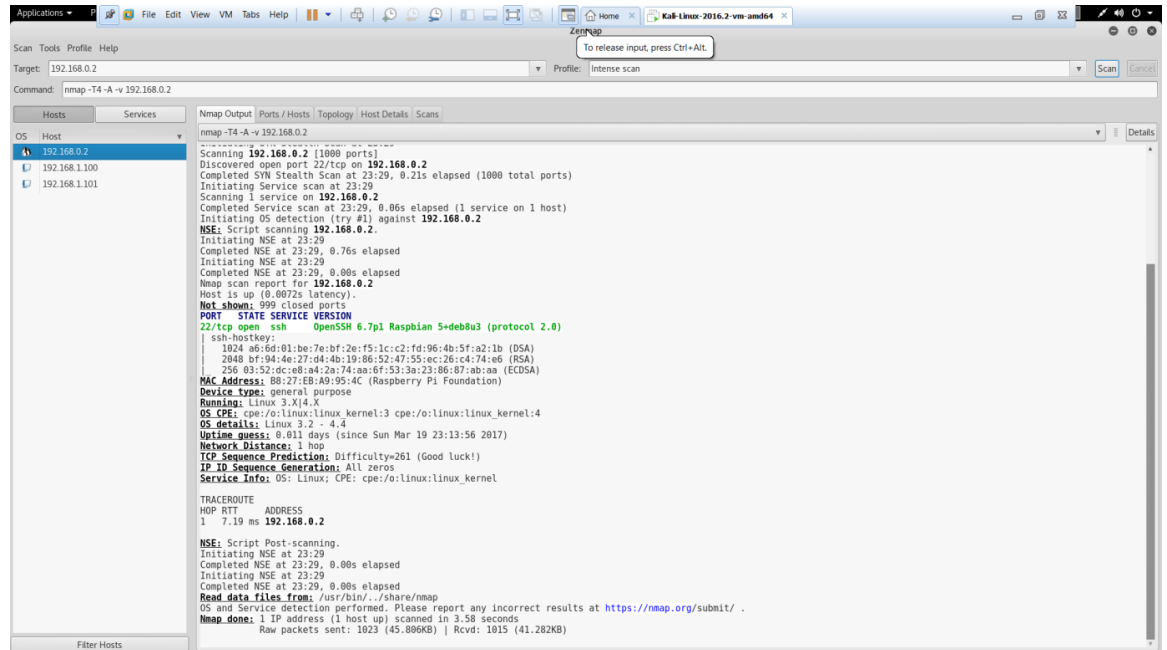


Fig 5.2: Testing using nmap

Comparison of Security Features Raspberry Pi & Home Router:

Home Router	Raspberry Pi		
Default	Default Rules	Custom Rules	Security Feature/Rules
Yes	Yes	Yes	Mac Filtering
Yes	Yes	Yes	Content Control
No	Yes	No	Shellcode
No	Yes	Yes	Virus
No	Yes	Yes	Network Scan

5.2 RESULT

Comparison of Raspberry Pi 3 & Home Router over Ethernet:

Home Router	Snort(Custom Rules)	Snort(Default Rules)	Snort off	
94.9	94.3	94.2	94.3	TCP (Mbps)
49.5	46.2	46.2	46.2	TCP Window 8K (Mbps)
1.07	1.05	1.05	1.05	UDP (Mbps)
0.430	0.459	0.579	0.374	Jitter (ms)
0/162	1/150	3/169	0/165	Loss

Comparison of Raspberry Pi 3 & Home Router Wirelessly :

Home Router	Snort(Custom Rules)	Snort(Default Rules)	Snort off	
26.0	18.2	13.2	19.2	TCP (Mbps)
17.6	12.7	10.6	12.9	TCP Window 8K (Mbps)
1.05	1.05	1.05	1.11	UDP (Mbps)
2.235	2.473	2.954	1.872	Jitter (ms)
0/159	1/161	1/160	0/170	Loss

6.1 Conclusion

The work carried out in this project is mainly focused to design and develop a proactive network surveillance framework to solve network security problem which is known to be difficult to circumvent. Though a significant amount of research has been done in the past to solve network security problems, but it still remains to be a challenging due to increased complexity and various threat vectors peeping into the network's daily. On the other hand, a lot of reactive techniques like firewalls, IDS are being adopted which cannot result into a complete solution. As a result, these tools and techniques proposed earlier may become obsolete for solving network security problems at large. Considering these facts, initially, an intensive study has been done and the findings have been reported in the thesis in the form of literature review. It was found that there are two kinds of approaches to solve network security problem: reactive and proactive. These security tools and techniques need to be woven in such a way that they produce positive sense of security amongst the security community. As reported in this research work, the layered approach to network security is best suited integrated solution which should be deployed in research organizations to learn more about new attacks and thus coming up with remedies for the same at the earliest. A domestic Home Router is not capable of analysing incoming and outgoing traffic. Thus lacks in providing security to the devices connected to the network. The above table depicts the same. Hence to make the home network more secure Raspberry Pi coupled with Snort is great alternate to a relatively insecure Home Router. This hardware can be configured precisely to the needs of the customer for better security solution which includes DOS and DDOS protection, anti-malware and antivirus alerts, content control and other numerous features as per requirements.

6.2 Future Scope

The study of IDS is quite new relative to many other areas of system research and it stands to reason that this topic offers number of opportunities for future exploration. Many improvements can be added to the IDS developed in this thesis.

- Developing an intrusion detection tool using method proposed in this project. Investigating applicability of unsupervised data mining techniques for intrusion detection.
- Developing a system that operates with a more global scope may be capable of detecting distributed attacks or those that affect an entire enclave.
- Development of such a system would be a valuable contribution to the study of intrusion detection.

7.1 REFERENCES

- [1] The Use of Computational Intelligence in Intrusion Detection Systems: A Review (2008) by Shelly Xiaonan, Wu Wolfgang Banzhaf, Shelly Xiaonan, Wu Wolfgang Banzhaf
<http://www.mun.ca/computerscience/research/MUN-CS-2008-05.pdf>
- [2] IDS on Raspberry Pi A Performance Evaluation by Andreas ASPERNÄS, Thommy SIMONSSON
<https://www.eecis.udel.edu/~cshen/367/Assignments/IDS-RPi.pdf>
- [3] Karthikeyan. K.R and A. Indra- “Intrusion Detection Tools and Techniques a Survey”
<http://www.ijcte.org/papers/260-G778.pdf>
- [4] INTRUSION DETECTION SYSTEM – A STUDY by Dr. S.Vijayarani and Ms. Maria Sylviaa.S <http://airccse.org/journal/ijsptm/papers/4115ijsptm04.pdf>
- [5] Asmaa Shaker Ashoor (Department computer science, Pune University) Prof. Sharad Gore (Head department statistic, Pune University),” Importance of Intrusion Detection System (IDS)”, International Journal of Scientific & Engineering Research, Vol. 2, Issue 1, Jan 2011. http://www.ijser.org/researchpaper%5CImportance_of_Intrusion_Detection_System.pdf
- [6] An Overview of Intrusion Detection System (IDS) along with its Commonly Used Techniques and Classifications by Hussain Ahmad Madni Uppal , Memoona Javed and M.J. Arshad, International Journal of Computer Science and Telecommunications [Volume 5, Issue 2, February 2014]
http://www.ijcst.org/Volume5/Issue2/p4_5_2.pdf
- [7] Intrusion Detection Data: Collection and Analysis by Robert F. Erbacher and Bill Augustine <https://pdfs.semanticscholar.org/4436/e1055b2d9915d69295d813c4ccabcb2957cb.pdf>
- [8] Survey and Taxonomy of Feature Selection Algorithms in Intrusion Detection System by You Chen, Yang Li, Xue-Qi Cheng, and Li Guo
<http://hiplab.mc.vanderbilt.edu/~y.chen/survey2006.pdf>

Screen Shots

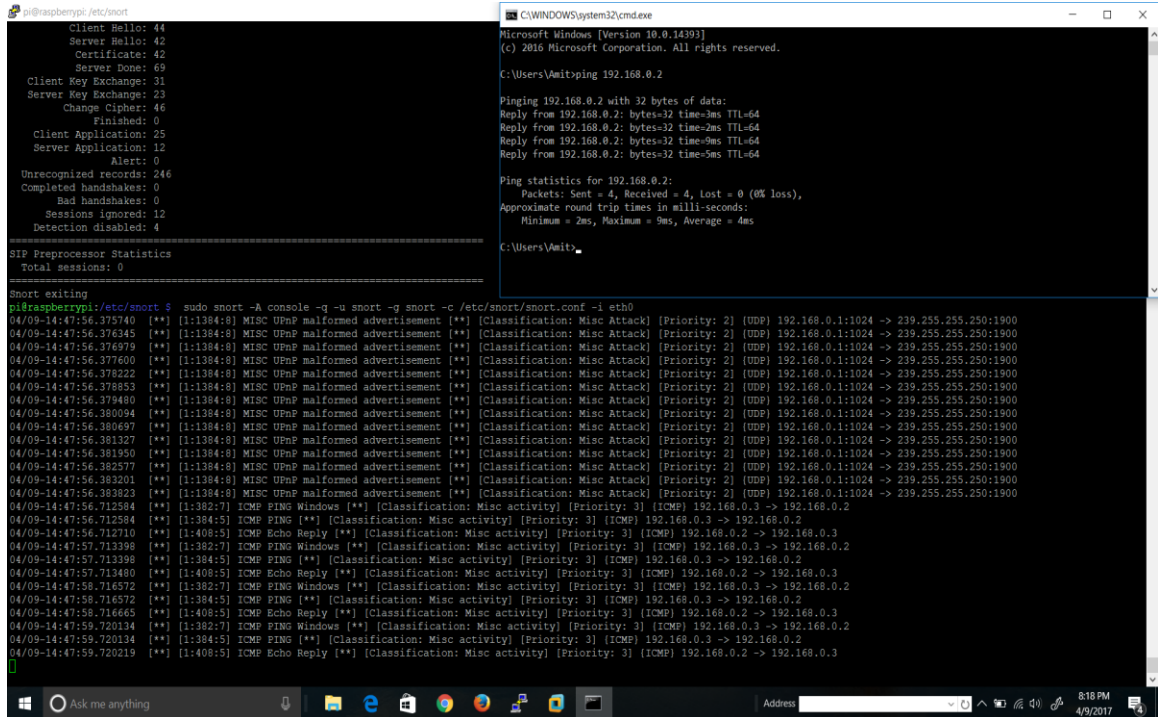


Fig 7.1: Pinging to Raspberry Pi

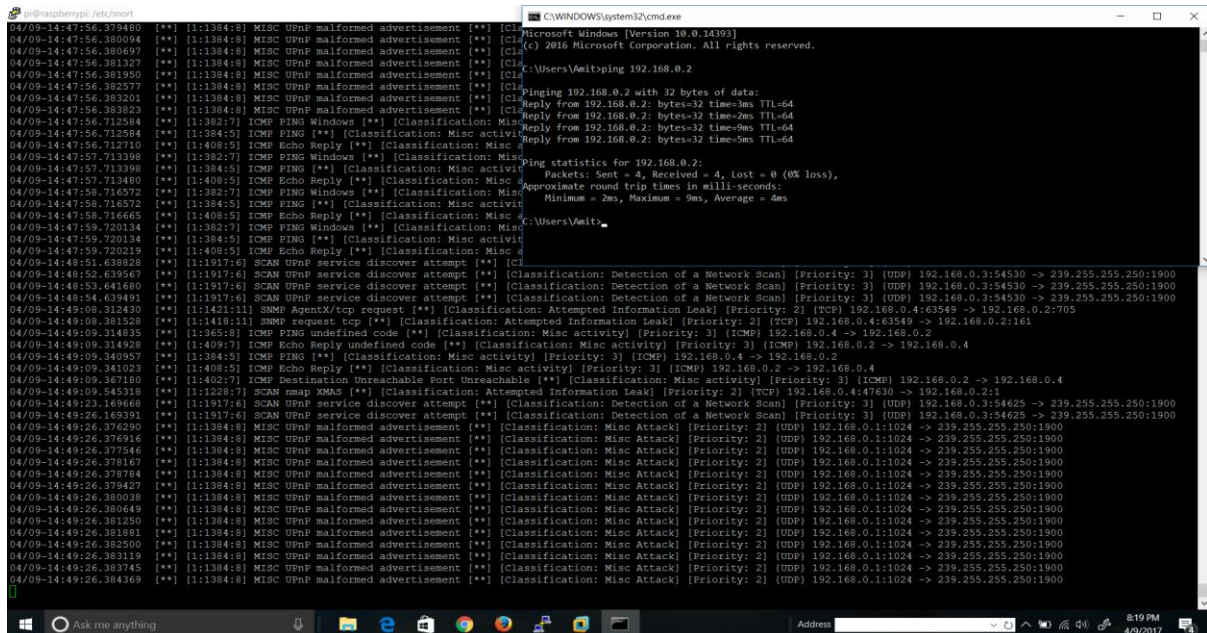


Fig 7.2: Alert on Snort Generated

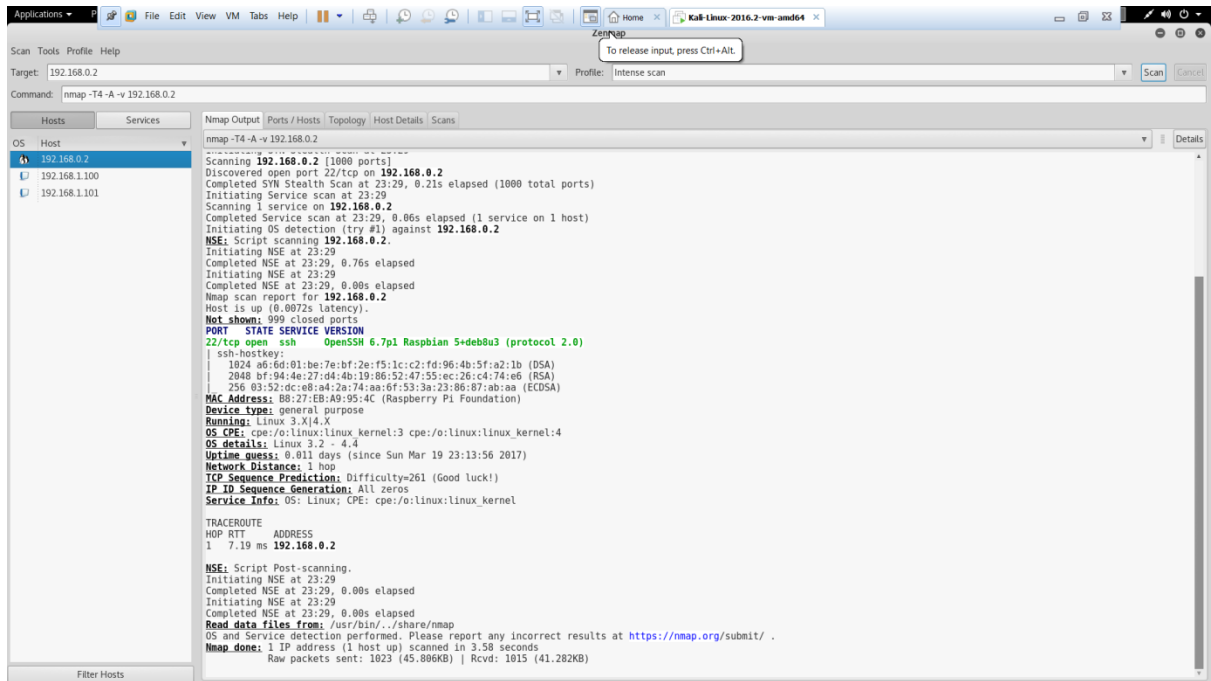


Fig 7.3: Scanning Available Ports using ZenMap

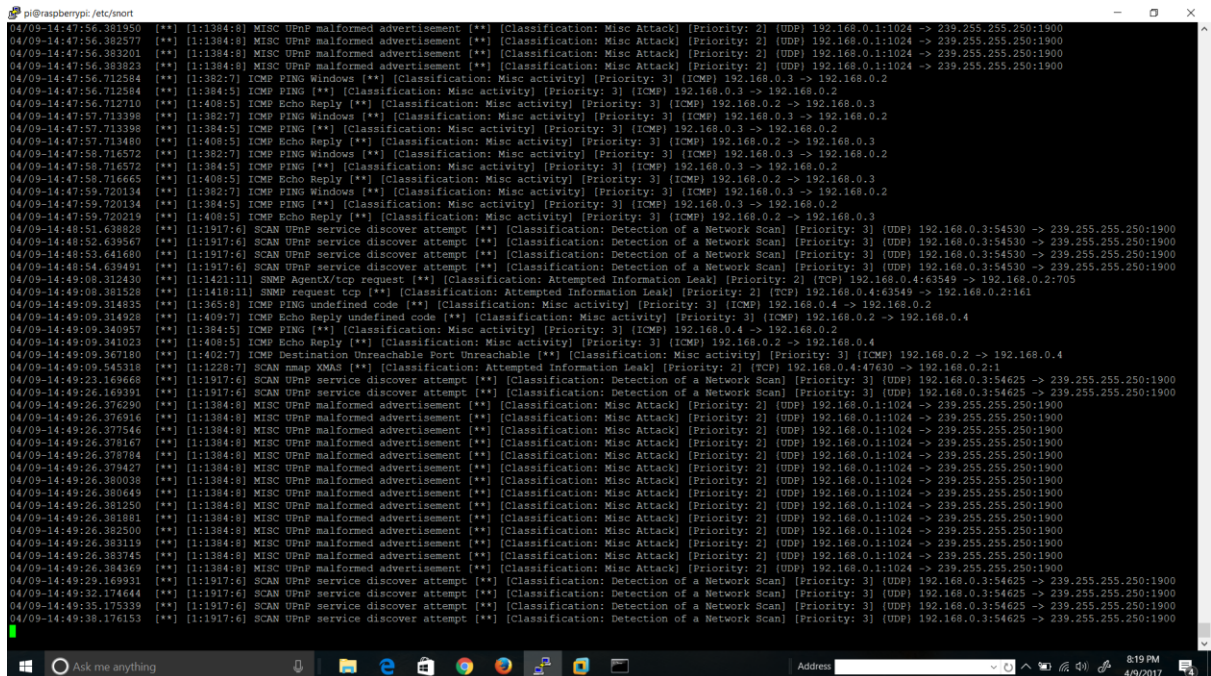


Fig 7.4: Snort alerting about the activity

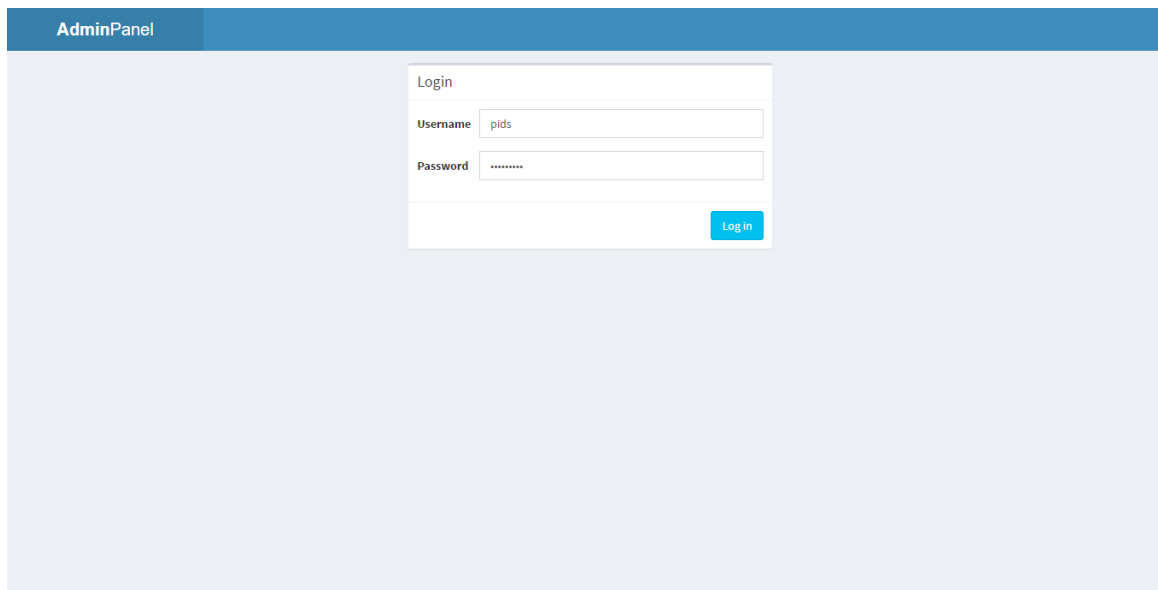


Fig 7.5: GUI for above System System

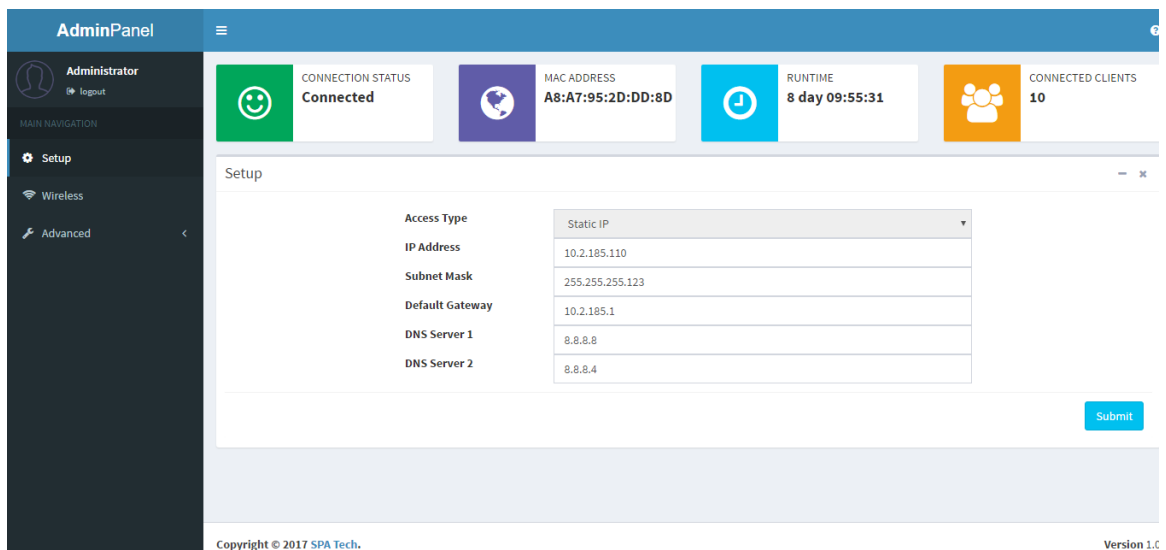


Fig 7.6: Network Setup

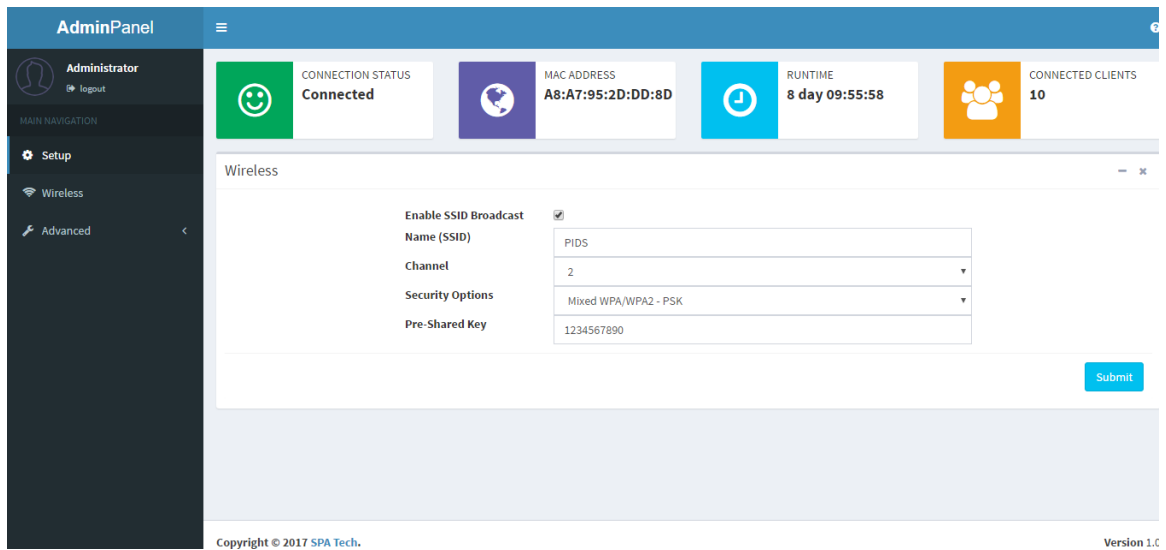


Fig 7.7 Wireless Setup

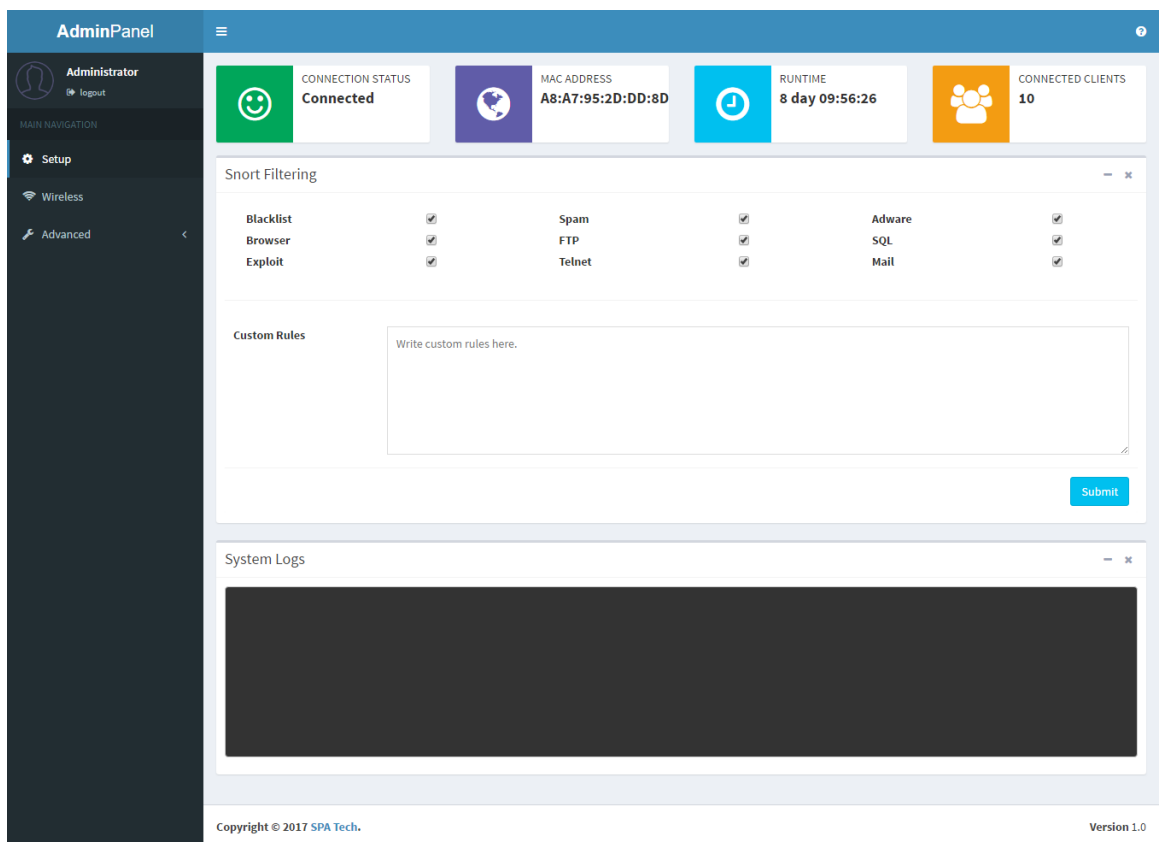


Fig 7.8 Selecting Snort rules

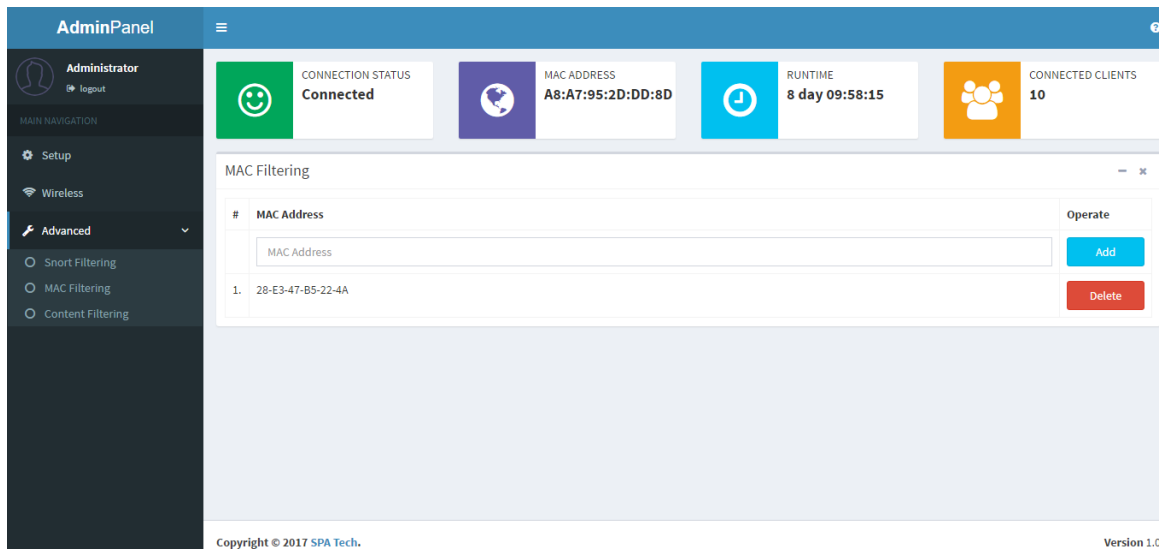


Fig 7.9 MAC Filtering

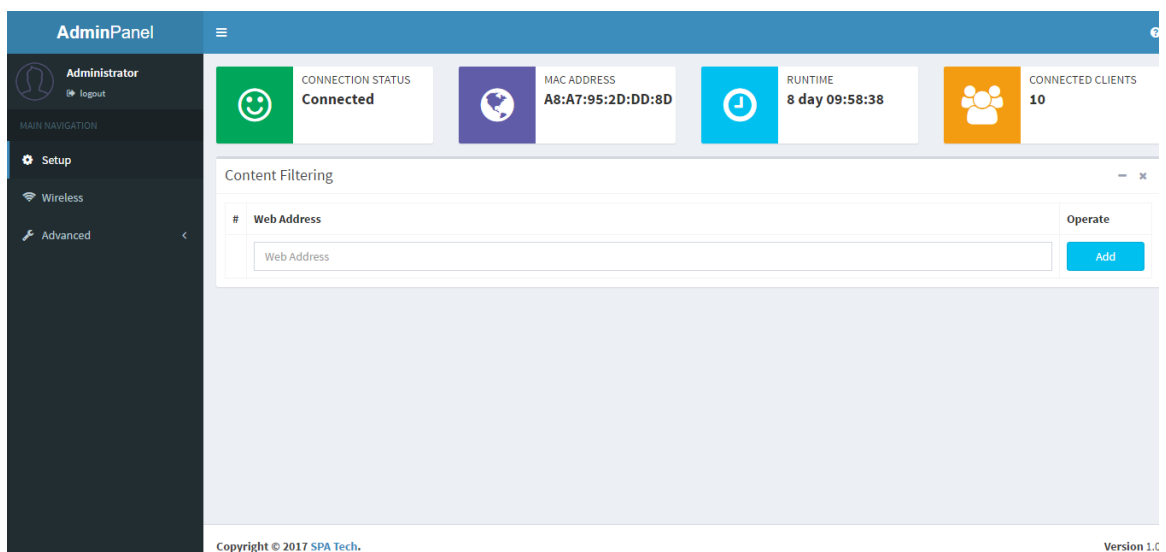


Fig 7.10 Web Content Filtering

Source Code

Forms.py

```
from django.contrib.auth.forms import AuthenticationForm
from django import forms

class LoginForm(AuthenticationForm):
    username = forms.CharField(label="Username", max_length=30,
    widget=forms.TextInput(attrs={'class': 'form-control', 'name': 'username'}))
    password = forms.CharField(label="Password", max_length=30,
    widget=forms.PasswordInput(attrs={'class': 'form-control', 'name': 'password'}))
```

views.py

```
import json
from django.shortcuts import render
from django.contrib.auth.decorators import login_required
from settings import CONFIG_URL
```

```
def index(request):
    context = {}
    return render(request, 'login.html')
```

```
@login_required(login_url="..")
def setup(request):
    if request.method == "POST":
        data = dict()
        data['IPAddress'] = request.POST.get("IPAddress")
        data['Subnet'] = request.POST.get("Subnet")
        data['Gateway'] = request.POST.get("Gateway")
        with open(CONFIG_URL+'setup.json', 'w') as outfile:
            json.dump(data, outfile)
```



```

with open(CONFIG_URL+'setup.json') as data_file:
context = json.load(data_file)
with open(CONFIG_URL+'status.json') as data_file:
context['status'] = json.load(data_file)
return render(request, 'setup.html', context)

```

```

@login_required(login_url="..")
def wireless(request):
printrequest.method
if request.method == "POST":
data = dict()
data['Broadcast'] = request.POST.get("Broadcast")
data['SSID'] = request.POST.get("SSID")
data['PreSharedKey'] = request.POST.get("PreSharedKey")
with open(CONFIG_URL+'wireless.json', 'w') as outfile:
json.dump(data, outfile)
with open(CONFIG_URL+'wireless.json') as data_file:
context = json.load(data_file)
with open(CONFIG_URL+'status.json') as data_file:
context['status'] = json.load(data_file)
return render(request, 'wireless.html', context)

```

```

@login_required(login_url="..")
def snort_filtering(request):
printrequest.method
if request.method == "POST":
data = dict()
data['label1'] = request.POST.get("label1")
data['label2'] = request.POST.get("label2")
data['label3'] = request.POST.get("label3")
data['label4'] = request.POST.get("label4")

```

```

data['label5'] = request.POST.get("label5")
data['label6'] = request.POST.get("label6")
data['label7'] = request.POST.get("label7")
data['label8'] = request.POST.get("label8")
data['label9'] = request.POST.get("label9")
data['label10'] = request.POST.get("label10")
data['customRules'] = request.POST.get("customRules")
with open(CONFIG_URL+'snortFiltering.json', 'w') as outfile:
    json.dump(data, outfile)
with open(CONFIG_URL+'snortFiltering.json') as data_file:
    context = json.load(data_file)
data_file = open(CONFIG_URL+'system.log', 'r')
context['logs'] = data_file.read()
with open(CONFIG_URL+'status.json') as data_file:
    context['status'] = json.load(data_file)
return render(request, 'snortFiltering.html', context)

```

```

@login_required(login_url="..")
defmac_filtering(request):
    context = {}
    return render(request, 'macFiltering.html')

```

```

@login_required(login_url="..")
defcontent_filtering(request):
    context = {}
    return render(request, 'contentFiltering.html')

```

urls.py

```

"""AdminPanel URL Configuration

```

The `urlpatterns` list routes URLs to views. For more information please see:

<https://docs.djangoproject.com/en/1.10/topics/http/urls/>

Examples:

Function views

1. Add an import: `from my_app import views`
2. Add a URL to `urlpatterns`: `url(r'^$', views.home, name='home')`

Class-based views

1. Add an import: `from other_app.views import Home`
2. Add a URL to `urlpatterns`: `url(r'^$', Home.as_view(), name='home')`

Including another URLconf

1. Import the `include()` function: `from django.conf.urls import url, include`
2. Add a URL to `urlpatterns`: `url(r'^blog/', include('blog.urls'))`

.....

```
from django.conf.urls import url
```

```
from django.contrib import admin
```

```
from django.contrib.auth import views as auth_views
```

```
from . import views, forms, main
```

```
from forms import LoginForm
```

```
from main import save_configs
```

```
urlpatterns = [
```

```
    # url(r'^$', views.index, name='index'),
```

```
    url(r'^setup/$', views.setup, name='setup'),
```

```
    url(r'^macFiltering/$', views.mac_filtering, name='mac-filtering'),
```

```
    url(r'^contentFiltering/$', views.content_filtering, name='content-filtering'),
```

```
    url(r'^$', auth_views.login, {'template_name': 'login.html', 'authentication_form': LoginForm},  
        name='login'),
```

```
    url(r'^logout/$', auth_views.logout, {'next_page': '/'}, name='logout'),
```

```
    url(r'^saveconfigs/$', main.save_configs, name='save-configs')
```

```
    # url(r'^login/$', djangoview.login, {'template_name': 'login.html', 'authentication_form':  
    LoginForm}),
```

```
]
```

```
MAIN.PY
```

```
from django.http import HttpResponse
```

```
from django.contrib.auth.decorators import login_required
```

```

import json
from settings import CONFIG_URL
from django.shortcuts import redirect

class Response():
    def __init__(self):
        self.err = False
        self.responseMsg = ""
        self.errMsg = ""
        self.responseData = []

    def save_setup_configs(request):
        data = dict()
        data['IPAddress'] = request.POST.get("IPAddress")
        data['Subnet'] = request.POST.get("Subnet")
        data['Gateway'] = request.POST.get("Gateway")
        data['DNS1'] = request.POST.get("DNS1")
        data['DNS2'] = request.POST.get("DNS2")
        with open(CONFIG_URL+'setup.json', 'w') as outfile:
            json.dump(data, outfile)

    @login_required(login_url="..")
    def try:
        params = request.POST.get("configName")
        if params == "setup":
            save_setup_configs(request)
            redirect('/') + params
            # print request.POST.get("title", "")
        responseObject['successMsg'] = 'Configuration saved successfully'
        responseObject['error'] = False
        response = HttpResponseRedirect(json.dumps(responseObject))
    except:
        responseObject['successMsg'] = 'Failed to save configurations'

```

```

responseObject['error'] = True

response = HttpResponse(json.dumps(responseObject))

return response

```

MAC_FILTERING.HTML

```

{% extends "index.html" %}

{% block content %}

<!--Mac Filtering -->

<div class="box">

<div class="box-header with-border">

<h3 class="box-title">MAC Filtering</h3>


<div class="box-tools pull-right">

<button type="button" class="btn btn-box-tool" data-widget="collapse"><i class="fa fa-minus"></i></button>

<button type="button" class="btn btn-box-tool" data-widget="remove"><i class="fa fa-times"></i></button>

</div>

</div>

<!-- /.box-header -->

<div class="box-body">

<table class="table table-bordered col-sm-12">

<tbody>

<tr>

<th style="width: 10px">#</th>

<th>MAC Address</th>

<th style="width: 10%">Operate</th>

</tr>

<tr>

<td></td>

<td><input type="text" class="form-control" placeholder="MAC Address"></td>

<td>

<button type="button" style="width: 100%" class="btn btn-info">Add</button>

```

```

</td>
</tr>
<tr>
<td>1.</td>
<td>00:00:00:a1:2b:cc</td>
<td>
<button type="button" style="width: 100%" class="btn btn-danger">Delete</button>
</td>
</tr>
</tbody>
</table>
<!-- /.col -->
</div>
<!-- /.box-body -->
</div>
<!-- /.box -->
</div>
<!-- /.col -->
</div>
<!--./MAC Filtering -->

{% endblock %}

```

SETUP.HTML

```

{% extends "index.html" %}
{% block content %}
<!--Setup-->
<div class="row">
<div class="col-md-12">
<div class="box">
<form action="../setup/" method="post">
    {% csrf_token %}
<div class="box-header with-border">

```

```

<h3 class="box-title">Setup</h3>
<div class="box-tools pull-right">
<button type="button" class="btn btn-box-tool" data-widget="collapse"><i class="fa fa-minus"></i>
</button>
<button type="button" class="btn btn-box-tool" data-widget="remove"><i class="fa fa-times"></i></button>
</div>
</div>
<!-- /.box-header -->
<div class="box-body">
<div class="row">
<div class="col-md-8" style="float: none;margin: auto">
<div class="box-body">
<div class="form-group">
<label for="selectAccessType" class="col-sm-3 control-label">Access Type</label>
<div class="col-sm-9">
<select class="form-control" id="selectAccessType" disabled>
<option>Static IP</option>
<option>Dynamic IP</option>
</select>
</div>
</div>
<div class="form-group">
<label for="inputIPAddress" class="col-sm-3 control-label">IP Address</label>
<div class="col-sm-9">
<input type="text" name="IPAddress" class="form-control" id="inputIPAddress"
placeholder="IP Address" value="{ { IPAddress } }">
</div>
</div>
<div class="form-group">
<label for="inputSubnet" class="col-sm-3 control-label">Subnet Mask</label>

```

```

<div class="col-sm-9">

<input type="text" name="Subnet" class="form-control" id="inputSubnet" placeholder="Subnet
Mask" value="{{ Subnet }}">

</div>

</div>

<div class="form-group">

<label for="inputGateway" class="col-sm-3 control-label">Default Gateway</label>

<div class="col-sm-9">

<input type="text" name="DNS1" class="form-control" id="inputDNS1" placeholder="DNS
Server 1" value="{{ DNS1 }}">

</div>

</div>

<div class="form-group">

<label for="inputDNS2" class="col-sm-3 control-label">DNS Server 2</label>

<div class="col-sm-9">

<input type="text" name="DNS2" class="form-control" id="inputDNS2" placeholder="DNS
Server 2" value="{{ DNS2 }}">

</div>

</div>

</div>

</div>

</div>

<!-- /.col -->

</div>

<div class="box-footer">

<button type="submit" class="btn btn-default">Cancel</button>

<button type="submit" class="btn btn-info pull-right">Submit</button>

</div>

<!-- /.row -->

</div>

</form>

<!-- /.box-body -->

</div>

<!-- /.box -->

```



```

</div>
<!-- /.col -->
</div>
<!--./Setup-->
{% endblock %}
SNORT FILTERING
{% extends "index.html" %}
{% block content %}

<!--Snort Filtering-->
<div class="row">
<div class="col-md-12">
<div class="box">
<form action="..../snortFiltering/" method="post">
    {% csrf_token %}
<div class="box-header with-border">
<h3 class="box-title">Snort Filtering</h3>

<div class="box-tools pull-right">
<button type="button" class="btn btn-box-tool" data-widget="remove"><i class="fa fa-
times"></i></button>
</div>
</div>
<!-- /.box-header -->
<div class="box-body">
<div class="row">
<div class="col-sm-4">
<div class="box-body">
<div class="form-group">
<label class="col-sm-8 control-label">label 1</label>
<div class="col-sm-4">
<input type="checkbox" name="label1" {% if label1 != null %} checked {% endif %} >
</div>
</div>

```

```

<div class="form-group">
<label class="col-sm-8 control-label">label 2</label>
<div class="col-sm-4">
<input type="checkbox" name="label2" {% if label2 != null %} checked {% endif %} >
</div>
</div>

<div class="form-group">
<label class="col-sm-8 control-label">label 3</label>
<div class="col-sm-4">
<input type="checkbox" name="label3" {% if label3 != null %} checked {% endif %} >
</div>
</div>

<div class="form-group">
<label class="col-sm-8 control-label">label 4</label>
<div class="col-sm-4">
<input type="checkbox" name="label4" {% if label4 != null %} checked {% endif %} >
</div>
</div>
</div>
</div>

<div class="col-sm-4">
<div class="box-body">
<div class="form-group">
<label class="col-sm-8 control-label">label 5</label>
<div class="col-sm-4">
<input type="checkbox" name="label5" {% if label5 != null %} checked {% endif %} >
</div>
</div>
</div>

<div class="form-group">
<label class="col-sm-8 control-label">label 6</label>
<div class="col-sm-4">
<input type="checkbox" name="label6" {% if label6 != null %} checked {% endif %} >
</div>

```

```

</div>
<div class="form-group">
<label class="col-sm-8 control-label">label 7</label>
<div class="col-sm-4">
<input type="checkbox" name="label7" {% if label7 != null %} checked {% endif %} >
</div>
</div>
<div class="form-group">
<label class="col-sm-8 control-label">label 8</label>
<div class="col-sm-4">
<input type="checkbox" name="label8" {% if label8 != null %} checked {% endif %} >
</div>
</div>
</div>
</div>
<div class="col-sm-4">
<div class="box-body">
<label class="col-sm-8 control-label">label 9</label>
<div class="col-sm-4">
<input type="checkbox" name="label9" {% if label9 != null %} checked {% endif %} >
</div>
</div>
<div class="form-group">
<label class="col-sm-8 control-label">label 10</label>
<div class="col-sm-4">
<input type="checkbox" name="label10" {% if label10 != null %} checked {% endif %} >
</div>
</div>
<div class="form-group">
<label class="col-sm-8 control-label">label 11</label>
<div class="col-sm-4">
<input type="checkbox" name="label11" {% if label11 != null %} checked {% endif %} >
</div>

```

```

</div>
<div class="form-group">
<label class="col-sm-8 control-label">label 12</label>
<input type="checkbox" name="label12" {% if label12 != null %} checked {% endif %} >
</div>
</div>
</div>
</div>
</div>
<!-- /.col -->
</div>
<div class="box-header with-border"></div>
<br>
<div class="row">
<div class="box-body">
<div class="form-group">
<label class="col-sm-2 control-label">Custom Rules</label>
<div class="col-sm-10">
<textarea name="customRules" class="textarea" style="width: 100%; height: 200px; font-size:
14px; line-height: 18px; border: 1px solid rgb(221, 221, 221); padding: 10px;" placeholder="Write
custom rules here.">{{ customRules }}</textarea>
</div>
</div>
</div>
</div>
</div>
<div class="box-footer">
<button type="submit" class="btn btn-default">Cancel</button>
<button type="submit" class="btn btn-info pull-right">Submit</button>
</div>
<!-- /.row -->
</div>
<!-- /.box-body -->
</form>
</div>
<!-- /.box -->

```

```

</div>
<!-- /.col -->
</div>
<div class="row">
<div class="col-md-12">
<div class="box">
<div class="box-header with-border">
<h3 class="box-title">System Logs</h3>

<div class="box-tools pull-right">
<button type="button" class="btn btn-box-tool" data-widget="collapse"><i class="fa fa-minus"></i>
</button>
<button type="button" class="btn btn-box-tool" data-widget="remove"><i class="fa fa-times"></i></button>
</div>
</div>
<!-- /.box-header -->
<div class="box-body">
<pre class="hierarchy bring-up"><code class="language-bash" data-lang="bash">{{ logs
}}</code></pre>
</div>
<!-- /.box-body -->
</div>
<!-- /.box -->
</div>
<!-- /.col -->
</div>
<!--./Snort Filtering -->

{% endblock %}
WIRELESS..HTML
{% extends "index.html" %}
{% block content %}

```

```

<!--Wireless-->
<div class="row">
<div class="col-md-12">
<div class="box">
<form action=" ../wireless/" method="post">
    {% csrf_token %}
<div class="box-header with-border">
<h3 class="box-title">Wireless</h3>

<div class="box-tools pull-right">
<button type="button" class="btn btn-box-tool" data-widget="collapse"><i class="fa fa-minus"></i>
</button>
<button type="button" class="btn btn-box-tool" data-widget="remove"><i class="fa fa-times"></i></button>
</div>
</div>
<!-- /.box-header -->
<div class="box-body">
<div class="row">
<div class="col-md-8" style="float: none;margin: auto">
<div class="box-body">
<div class="form-group">
<label for="checkBroadcast" class="col-sm-3 control-label">Enable SSID Broadcast</label>
<div class="col-sm-9">
<input name="Broadcast" type="checkbox" id="checkBroadcast" {% if Broadcast != null %}
checked {% endif %} >
</div>
</div>
<div class="form-group" style="clear: both">
<label for="inputSSID" class="col-sm-3 control-label">Name (SSID)</label>

<div class="col-sm-9">

```

```

<input name="SSID" type="text" class="form-control" id="inputSSID" placeholder="SSID"
value="{{ SSID }}">

</div>

</div>

<div class="form-group">
<label for="selectChannel" class="col-sm-3 control-label">Channel</label>

<div class="col-sm-9">
<select class="form-control" id="selectChannel" name="Channel">
<option {% if Channel == "Auto" %} selected="selected" {% endif %}>Auto</option>
<option {% if Channel == "1" %} selected="selected" {% endif %}>1</option>
<option {% if Channel == "2" %} selected="selected" {% endif %}>2</option>
<option {% if Channel == "3" %} selected="selected" {% endif %}>3</option>
<option {% if Channel == "4" %} selected="selected" {% endif %}>4</option>
<option {% if Channel == "5" %} selected="selected" {% endif %}>5</option>
<option {% if Channel == "6" %} selected="selected" {% endif %}>6</option>
<option {% if Channel == "7" %} selected="selected" {% endif %}>7</option>
<option {% if Channel == "8" %} selected="selected" {% endif %}>8</option>
<option {% if Channel == "9" %} selected="selected" {% endif %}>9</option>
<option {% if Channel == "10" %} selected="selected" {% endif %}>10</option>
</select>
</div>
</div>

<div class="form-group">
<label for="selectSecurity" class="col-sm-3 control-label">Security Options</label>

<div class="col-sm-9">
<select class="form-control" id="selectSecurity" name="Security">
<option {% if Security == "Open" %} selected="selected" {% endif %}>Open</option>
<option {% if Security == "Shared" %} selected="selected" {% endif %}>Shared</option>
<option {% if Security == "Mixed WPA/WPA2 - PSK" %} selected="selected" {% endif %}>Mixed
WPA/WPA2 - PSK</option>
<option {% if Security == "WPA/WPA2-Enterprise" %} selected="selected" {% endif
%}>WPA/WPA2-Enterprise</option>

```

```

</select>
</div>
</div>
<div class="form-group">
<label for="inputPreSharedKey" class="col-sm-3 control-label">Pre-Shared Key</label>

<div class="col-sm-9">

<input type="text" name="PreSharedKey" class="form-control" id="inputPreSharedKey"
placeholder="Pre-Shared Key" value="{{ PreSharedKey }}">
</div>
</div>
</div>
</div>
</div>
<!-- /.col -->
</div>
<br />
<div class="row">
<div class="form-group" style="float: none;margin: auto">
<div class="col-sm-2" style="float: none;margin: auto">
<input type="submit" class="btn btn-block btn-primary btn-sm" id="submitSetup"
value="Submit">
</div>
</div>
</div>
<br />
<!-- /.box -->
</div>
<!-- /.col -->
</div>
<!--./Wireless-->
{% endblock %}

```