Design and Development of Policy Scripts to Detect Network Intrusions Using Bro

Thesis submitted in partial fulfillment of the requirements for the award of degree of

Master of Engineering
in
Software Engineering

Thapar University, Patiala

By:
Sanmeet Kaur
(80631020)

Under the supervision of:
Dr. Maninder Singh
Assistant Professor
CSED,TU,Patiala

JUNE 2008

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA – 147004

Certificate

I hereby certify that the work which is being presented in the thesis entitled, **"Design and Development of Policy Scripts to detect Network Intrusions using Bro"**, in partial fulfillment of the requirements for the award of degree of Master of Engineering in Software Engineering submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Maninder Singh* and refers other researcher's works which are duly listed in the reference section.

The matter presented in this thesis has not been submitted for the award of any other degree of this or any other university.

(**Sanmeet Kaur)**

This is to certify that the above statement made by the candidate is correct and true to the best of my knowledge.

(**Dr. Maninder Singh)**
Assistant Professor
Computer Science and Engineering Department
Thapar University
Patiala

Countersigned by

(Dr. Seema Bawa)
Professor & Head,
Computer Science & Engineering Department,
Thapar University,
Patiala

(Dr. R.K. Sharma)
Dean(Academic Affairs),
Thapar University,
Patiala.

Acknowledgement

I would like to express my sincerest thanks to my thesis supervisor Dr. Maninder Singh, Assistant Professor, Computer Science and Engineering Department for his inspiration, guidance, stimulating suggestions, immense help and support throughout the period of this research work. He has provided me with all the necessary resources including motivation and research environment without which I could not complete this work. It was a great opportunity for me to do this work under his supervision.

I would like to thank Mr. Vishal Verma, Senior Software Engineer, Bivio Networks, for his great technical support and motivation. He has always been motivating and ready to help irrespective of his busy schedule.

I would like to thank Dr. Seema Bawa Professor and Head, Computer Science and Engineering Department for her moral support and the research environment she had facilitated for this work.

I would also like to thank all my Teachers for their stimulating discussion and invaluable support I received during the period of this research. I am thankful to all the Bro Team members who are regularly maintaining the mailing list and responding to the problems sent by me. I am also thankful to the authors whose works I have consulted and quoted in this work.

Finally, I wish to thank my dearest family and friends for all their immense love, enthusiastic encouragement and support throughout my life without which I could not complete this work. Last but not the least I would like to thank thy God who has always been with me in my good and bad times.

Sanmeet Kaur
(80631020)

Abstract

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. Many methods have been developed to secure the network infrastructure and communication over the Internet, among them the use of firewalls, encryption, and virtual private networks. Intrusion detection is a relatively new addition to such techniques. Intrusion detection methods started appearing in the last few years. 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, as well as for legal purposes. Both commercial and open source products are now available for this purpose. Many vulnerability assessment tools are also available in the market that can be used to assess different types of security holes present in your network**.** There are different techniques to detect Intrusions.

This work discusses various such techniques to detect intrusions. Also the Intrusion Detection System called Bro is discussed. The major emphasis is on the design and development of the policy scripts to detect various network intrusions. These scripts are written using the scripting language of Bro, which supports various special data types to support network level activities. It also has signature-matching features to make threat signatures to match against various attacks and detect them later.

Table of Contents

List of Figures

Figure No **Name of Figure** **Page No**

Chapter 1
Introduction

1.1 Introduction to Network Security

Computer networks have brought the world together by bridging the information gap among people. Network technology has undergone a revolution with better and faster ways of sending information between computers. Unfortunately security systems and policies to govern these networks have not progressed as rapidly. With the birth of the Internet in 1969, computers and computer networks that were isolated from each other were suddenly interconnected. The Internet has grown at an explosive rate with innovations in communication and information technologies. While the Internet has made it easier to reach and communicate with people all over the globe, it has also made it easier to attack computer systems connected to it. With e-commerce and online banking being more and more employed these days, security for systems offering these services has become necessary. Furthermore, the requirement for ease of communication has forced companies and government organizations to open their computer networks to the Internet. To add to this, more and more operating system flaws, and network protocol features that can be exploited, are being discovered with each passing day [1].

The past few years have seen a radical evolution in the nature and requirements of network security. There are many factors contributing to these changes, the most important of which is the shift in focus from so-called 'network-level' threats, such as connection-oriented intrusions and Denial of Service (DoS) attacks, to dynamic, content-based threats such as Viruses, Worms, Trojans, Spyware and Phishing that can spread quickly and indiscriminately, and require sophisticated levels of intelligence to detect. Where attacks like Smurf, Fraggle and the Ping of Death were the key threats in years past, now attacks such as "Microsoft IIS 5.0 printer ISAPI extension buffer overflow vulnerability" and "Unicode directory traversal" are more prevalent [7].

There are several major drivers that are shaping the new security landscape:

- Increasing complexity of networks
- Increasing sophistication of applications and attacks
- Financial rewards for hackers with the advent of Spyware and Phishing
- Security as a tool to increase workforce productivity

Where a network 10 years ago might have consisted of a LAN connected to the Internet through a WAN connection, and maybe a few remote access or site-to-site VPN tunnels, the reality today is much more complex. A common environment today will have multiple access mechanisms into the network, including 802.11 wireless LAN, web portals for partners and customers, FTP servers, email servers, end-users using new communication platforms and peer-to-peer applications for file-sharing. An example of such a network, and the threats that are present, is illustrated in Figure 1.1.

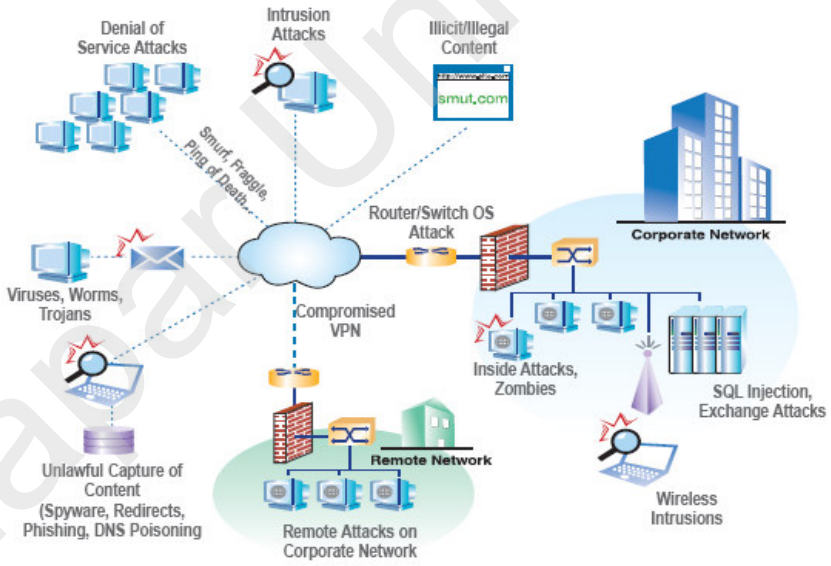


Figure 1.1: Major threats in today's network [7]

1.1.1 Need of Network Security

Computer security technology is still in its infancy. Technologies such as firewalls, antivirus, and IDS have migrated from research labs into production networks, and have become required mainstays both as essential defenses and as legally mandated compliance systems. Computer security systems are complex

devices that need to meet a variety of conflicting goals: high performance, fault tolerance, easy administration – and rigorous security processing. Some vendors have staked their claim based on speed, others on cost, and still others on the defensive posture and security of their products. Unfortunately, it is extremely difficult for the customer to sort through marketing fluff and dubious benchmarks, to determine which products actually work and which merely appear to work. Few customers are sufficiently sophisticated or willing to take the time to do their own testing and most are forced to rely on published results from trade magazines, recommendations from consultants, or industry analysts. Sadly, few of the trade magazines or analysts have the sophistication or time to perform adequate testing, either [5].

While using the Internet, along with the convenience and speed of access to information new risks arise. Among them are the risks that valuable information will be lost, stolen, corrupted, or misused and that the computer systems will be corrupted. If information is recorded electronically and is available on networked computers, it is more vulnerable than if the same information is printed on paper and locked in a file cabinet. Intruders do not need to enter an office or home, and may not even be in the same country. They can steal or tamper with information without touching a piece of paper or a photocopier. They can create new electronic files, run their own programs, and even hide all evidence of their unauthorized activity [27].

It is remarkably easy to gain unauthorized access to information in an insecure networked environment, and it is hard to catch the intruders. Even if users have nothing stored on their computer that they consider important, that computer can be a "weak link", allowing unauthorized access to the organization's systems and information. Information that intruders find useful includes which hardware and software are being used, system configuration, type of network connections, phone numbers, and access and authentication procedures. Security-related information can enable unauthorized individuals to get access to important files and programs, thus compromising the security of the whole system. Today's commercial off-the-shelf software technology is riddled with vulnerabilities. These include vulnerabilities that allowed viruses and worms and other manual and automated attacks to inflict damages costing hundreds of millions of dollars per occurrence. Specifically: LoveLetter, a worm that severely clogged mail servers and networks

in 2000; Code Red, an aggressive worm that attacked unpatched Microsoft web servers and defaced their main pages; and most recently, Nimda, a worm that spread by several different methods including email and web protocols, and searched for as many as 16 separate vulnerabilities to attack. The recent Distributed Denial of Service (DDOS) attacks are less serious but still expensive. Virus attacks, exploits directed at unpatched popular firewalls (e.g., Check Point, Cisco Pix), buffer overflows, directory traversal, other more obscure attacks against web servers, and the scope of the problem starts to become quite clear. Those affected include banks and financial companies, insurance companies, brokerage houses, consultants, government contractors, government agencies, hospitals, medical laboratories, network service providers, utility companies, the textile industry, universities, wholesale and retail trades. The consequences of a break-in cover a broad range of possibilities: a minor loss of time in recovering from the problem, a decrease in productivity, a significant loss of money or staff-hours, a devastating loss of credibility or market opportunity, a business no longer able to compete, legal liability, and the loss of life.

1.1.2 Various Threats to Network Security

The list is very long. But still we have tried to cover the major ones:

- **Scanning a network**

The first step in an attack is reconnaissance – finding out as much as possible about the target. Many tools are available for investigating a network – ranging from simple scripts to commercial network mapping tools, to dedicated scanning applications. In essence, these tools send packets to a potential host, and deduce information about that host from any reply. Mapping a network consists of checking every possible address for that host. In particular, a number of scan types can be distinguished.

- **Ping scan**: The simplest form of scan in which an attacker sends an ICMP echo request packet to every candidate machine (which is the same way the *ping* tool works). Any addresses that respond are noted as active [14].

- **TCP Connect() scan**: Another simple scan, an attacker attempts to open a standard TCP connection to a typical port on the candidate machine (such as the HTTP port 80). Any machine where such a connection succeeds is noted as active. Since many systems log any connection attempts, this type of scan is relatively easy to recognize from standard audit data.

- **TCP SYN(Stealth) scan**: This scan sends a connect request to every candidate machine (similar to the Connect() scan), but does not complete the connection by sending a final SYN/ACK packet. In this way, the connection fails and does not generally show up in the system logs – hence a "stealth" scan. Since this scan has a similar signature to a SYN flood attack, many security systems now log such occurrences.

- **Stealth FIN, Xmas, ACK and NULL scans**: These scans all form part of the same family of variations on the SYN scan techniques. Each sends a special packet to a candidate address, deducing whether a port is open or not from RST reply packets (which indicate a closed port). If not reply is received the port is open – or the request lost in transit, such as being discarded by a firewall. FIN scans consists of packets with the FIN flag set, Xmas scans of packets with the FIN, URG and PUSH flags set, and NULL scans of packets with no set flags. The ACK scan consists of packets with the ACK flag set (generally denoting replies), and so are often capable of penetrating firewalls.

- **UDP scans**: This scan consists of sending UDP packets to likely ports on candidate machines – at worst, scanning for any open UDP ports. Since UDP is connectionless, such attempts are for details on how these services function through firewalls and filters [14].

Many variations on these scanning techniques exists – including scans using fragmented packets, and scans spread across a long period or a number of source machines. In practice, completely blocking scans is probably infeasible – but may give an administrator early warning of an impending attack.

- **Buffer Overflows**

This is actually rich category of specific attacks, all using similar weaknesses in software. The core of the attack is to pass an unusually structured (often very long) value as a parameter to a system, when it is expecting something else – for example, requesting an FTP server to change the working directory to an extremely long filename. What happens, in general, is that the parameter overflows its storage buffer, overwriting commands that would later be executed – allowing an attacker to have arbitrary commands executed by the remote server. These commands can then be used to do any number of things – typically, creating an interactive shell, modifying access restrictions, or retrieving sensitive information, such as a password list.

- **Open doors and abused trust**.

In order to simplify authentication and access control, many systems accept assertions made by trusted systems. For example, the *rsh* series of commands accepts the remote machine's claims to user identity, if the remote machine is authorized to make such claims. This allows a number of attack techniques, based around abusing the assumptions made in such systems. One technique involves an attacker assuming the identity of a trusted machine, allowing it access to the trusting system. Another is based on the fact that under some systems (such as some UNIX variants), users can control which other machines are trusted (using the *.rhosts* file). A common escalation step in attacking such a host is to modify this file, to allow the attacker free access.

- **Social Engineering**.

This type of attack is one of the oldest and most effective ways of bypassing security mechanisms: fool somebody with the ability to do it for you. Variations range from guessing information based on the attacker's knowledge of the target involved, to impersonating personnel, and more. The only way to protect an organization is to ensure that it has a sufficiently clear security policy, and that its users are educated – no technical measures can prevent this type of attack.

- **Application Attacks**.

These attacks depend on convincing an application to do something it was not expected to – overwrite files, execute commands it should not, or give away information that should be hidden. In addition, these attacks are notable since they can often penetrate even the best developed security mechanisms – the only defense is to keep the applications themselves secure. Examples include requesting password files via FTP or HTTP, attempting to overwrite sensitive files via the same, or passing unexpected information to server applications – such as any of the range of CGI exploits available.

- **Trojan software**.

Trojan horses are computer programs that appear to be useful software, but instead they compromise the security and cause a lot of damage. A recent Trojan horse came in the form of an e-mail that included attachments claiming to be Microsoft security updates, but turned out to be viruses that attempted to disable antivirus and firewall software. Trojan horses spread when people are lured into opening a program because they think it comes from a legitimate source. Trojan horses can also be included in software that downloaded for free. Never download software from a source that is not trusted. The problem of computer viruses is well-known; but the techniques used for propagating these programs can also be used to compromise security. A good example is the Back Orifice system – once an infected application is run on a system, it installs a backdoor on the system, allowing the attacker free access . Preventing this type of attack is difficult – it requires user education, and security to be deeply embedded into systems [23].

- **Viruses**

A computer virus is a special kind of computer program which: Spreads across disks and networks by making copies of itself, usually surreptitiously and / or can produce undesired side-effects in computers in which it is active. Viruses (and Worms) are a class of attack whereby an infected attachment or download causes damage to a host system or network. The damage can range from minor (client DoS attack) to catastrophic (full-blown corruption of critical stored information or system registries). A critical trend that is resulting from the increased

sophistication of Viruses is the rapidly decreasing "window of infection". In July of 2001, it took the Code Red virus just under 6 hours to infect 359,000 clients. Just eighteen months later, the Slammer worm infected 75,000 clients in under 30 minutes. The threats are real and spread fast. Security vendors have responded by trying to decrease their own "windows of inoculation" which is the time it takes to detect a threat, issue a patch release, and download it to its host systems under management.

There is also a new class of virus-related attack called a 'blended threat'. A blended threat is a 'perfect attack' whereby a virus is accompanied by a number of other attack and intrusion techniques to maximize penetration and damage. SoBig and Sasser are good examples of how complicated it has become to detect and prevent sophisticated application-layer attacks.

- **Spyware**

Spyware (and Adware) is one of the most misunderstood of the new generation of application-layer threats because there is no consensus on what defines a threat (or more appropriately, what the difference is between 'annoying' Adware and a true threat). There are three general classes of Spyware:

- **Harmless-but-annoying**

  Generally consists of actions such as changing the default home page of your browser, or unsolicited/untargeted pop-up ads.

- **Information-collecting**

  Cookies are the most common type of information collecting mechanism, but simple keystroke and activity loggers are becoming more common. This class of Spyware is generally interested in collecting basic information about you, the sites you visit, and other preferences so that a 3rd party can send you targeted ads or promotions. There is generally not malicious intent, but many would call this an invasion of privacy.

- **Malicious**

  It generally includes full keystroke logging and collecting private information with the intent of sending the information to a collection server. The

information is collected, and sold to 3rd parties who have varying interests. Even today, this type of Spyware can be downloaded instantly on a Client device simply by visiting a URL, no further clicking necessary. This type of Spyware is illegal and critical for an organization to detect and stop.

1.1.3 The Security Wheel

This industry-standard model has been chosen to illustrate where security management fits in, and how all security activities in a network must evolve around the security policy; the concept sees network security as a continuing process built around a corporate security policy. This process is divided into the stages:

- Implement network security
- Monitor network and respond to incidents
- Test the security of the network
- Improve network security.

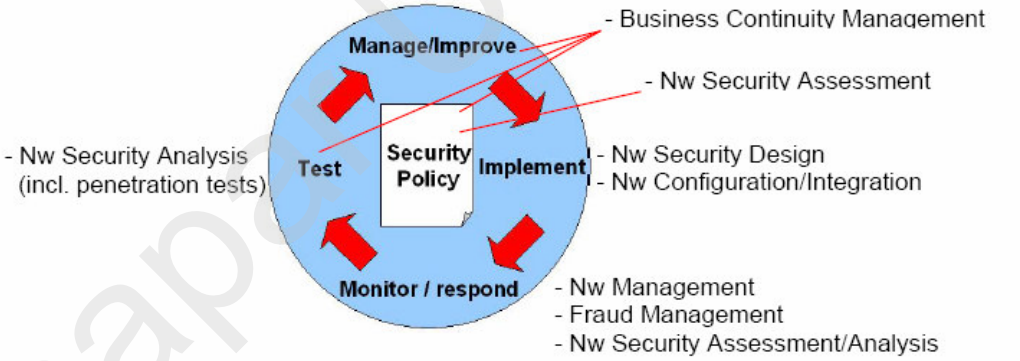


Figure 1.2: Security Wheel: Security as a Continuous Process [6]

Implement network security – Security devices such as perimeter nodes, VPN devices, firewalls, Intrusion /Prevention Systems (IDS/IPS) and authentication devices are planned, configured and integrated. The purpose is to prevent activities that the policy has defined as threats.

Monitor/Respond – The implemented security policy is validated using intrusion detection, as well as log and other auditing techniques, to watch for violations.

Test – The effectiveness of the policy should be evaluated at regular intervals through security audits, vulnerability scanning and/or penetration tests.

Manage/Improve – Information gathered from previous steps is analyzed and used together with developments in the security market to improve the policy, moving around the circle to the first step again [6].

1.2 Introduction to Intrusion Detection

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. Many methods have been developed to secure the network infrastructure and communication over the Internet, among them the use of firewalls, encryption, and virtual private networks. Intrusion detection is a relatively new addition to such techniques. Intrusion detection methods started appearing in the last few years. Using intrusion detection methods, one 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, as well as for legal purposes. Both commercial and open source products are now available for this purpose. Many vulnerability assessment tools are also available in the market that can be used to assess different types of security holes present in your network. A comprehensive security system consists of multiple tools, including:

- Firewalls that are used to block unwanted incoming as well as outgoing traffic of data. There is a range of firewall products available in the market both in Open Source and commercial products.
- Intrusion detection systems (IDS) that are used to find out if someone has gotten into or is trying to get into your network.
- Vulnerability assessment tools that are used to find and plug security holes present in your network. Information collected from vulnerability assessment tools is used to set rules on firewalls so that these security holes are safeguarded from malicious Internet users. There are many vulnerability assessment tools including Nmap and Nessus [23].

1.2.1 Intruder's Definition

The common term for an intruder is a "hacker" or programmer who tries to break in and enter a computer system with malicious intent. Another category of intruders includes persons within a system misusing privileges to gain access to information and systems they are unauthorized to use [12].

1.2.2 Types of Intruders

Intruders are basically of two types: external and internal. External intruders do not have any authorized access to the system they attack. Internal intruders on the other hand have some access rights, but they seek to gain additional capability. A surprisingly large number of security breaches are due to internal intruders. Their motivation to hack is either because they can hack, to cause destruction if they are disgruntled employees, or to profit from the company [1].

1.2.3 Detecting Intrusion

An Intrusion is a deliberate unauthorized attempt to access information, manipulate information, and/or commit acts to render a system unreliable or unusable. Intrusion detection is performed through two main checks – Anomaly Detection and Signature Recognition.

- **Checking Signature Rules**

This is an exercise where the IDS looks for a "signature" or pattern. IDSs contain databases with strings based on patterns for known hacker techniques. IDSs examine the network traffic, looking for well-known patterns of attack.

- **Detecting Anomalies**

This is by identifying statistical anomalies with reference to a baseline of statistics on activities like CPU utilization, user logins, file activity, disk activity, and so on. When there is an abnormal or unusually high or low value from the baseline, the IDS is able to detect it [12].

1.2.4 Architecture of Intrusion Detection System

Intrusion Detection systems have evolved from monolithic batch-oriented structures to complex, distributed real-time networks of components. In this development, a basic general model has emerged, allowing discrete functional

components to be distinguished. A typical IDS structure consists of the following components:

- Sensors
- Monitors
- Resolver
- Controller

Sensors: These are the data-gathering component of IDS. These are similar monitoring processes on networked hosts those extract information from the hosts' event logs, audit information, application logs, and general state. They may also be dedicated network monitors connected to an observation point on a network segment. From there, a network monitor inspects all visible network traffic, synthesizing event logs from observed traffic. These systems also filter the event logs, generating summaries that are forwarded to IDS Monitors.

Monitors: These are the processing segment of IDS. These systems receive and interpret event summaries received from sensors. These event summaries are then inspected for anomalous or suspicious activity, and suspicion reports are generated. The suspicion reports are forwarded to higher level monitors, or to resolver units.

Resolver: These modules receive suspicion reports from monitors, and are responsible for determining appropriate responses like reporting to an administrator, changing the behaviour of lower level components or reconfiguring other security mechanisms such as firewalls.

Controller: In distributed IDS, configuration of components is possible via centralized controllers. These modules simplify administration, and allow administrative personnel to rapidly reconfigure IDS components for example, increasing the records kept in the case of an intrusion. These modules however are not involved in the ongoing functioning of IDS.

The division between these modules is often indistinct. Early, single-system IDS had all four components functioning as a single unit. With the development of

distributed IDS, however, these components are becoming more distinct. Systems such as GrIDS, AAFID, and EMERALD extend this model by applying these components in a cascading fashion – allowing higher level system overviews to be gained as a user ascends through the tree [23].

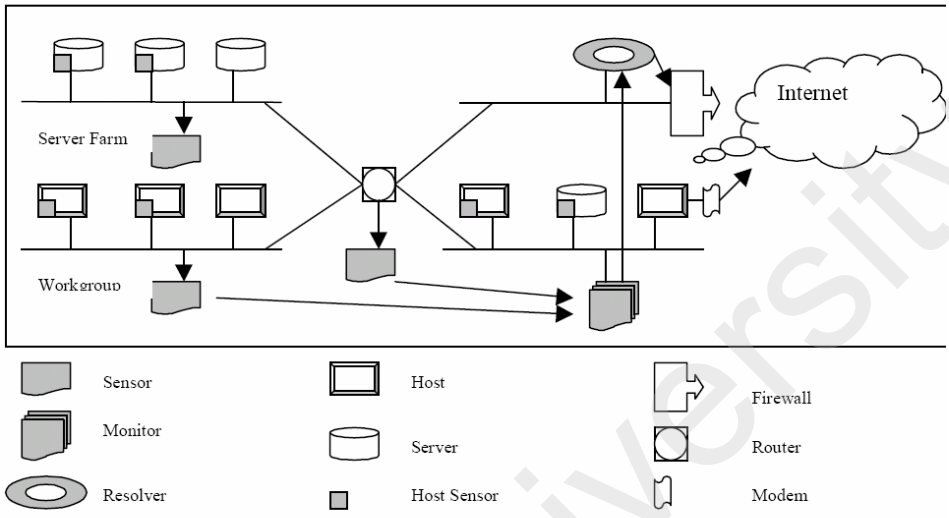


Figure 1.3: Intrusion Detection System Networks [23]

1.2.5 The Need for IDS

When most people think of network security, they think *Firewall*. Firewalls are widely deployed as a first level of protection in a multi-layer security architecture, primarily acting as an access control device by permitting specific protocols (such as HTTP, DNS, SMTP) to pass between a set of source and destination addresses. Integral to access policy enforcement, firewalls usually inspect data-packet headers to make traffic-flow decisions. In general, they do not inspect the entire content of the packet and can't detect malicious code embedded within normal traffic. It should be noted that routers also offer some rudimentary protection through packet-filtering processes. While firewalls and router-based packet filtering are necessary components of an overall network security topology, they are insufficient on their own. Network IDS products inspect the entire content of every packet traversing the network to detect malicious activity. This content inspection technique provides deeper packet analysis as compared to a firewall or a router. Intrusion Detection Systems are effective when sophisticated attacks are embedded in familiar protocols, such as an HTTP session, which would normally

pass undetected by a firewall. It's not surprising that the processing power required for an Intrusion Detection System is an order of magnitude higher, when compared to a firewall product. Permeable modern networks have made IDS products essential tools as security engineers strive to detect, analyze, and protect networks against malicious attack. As a result, IDS products are being deployed outside and inside firewalls and are quickly becoming mainstays in *best practice* secure network implementations.

1.2.6 IDS in Network Topology

Depending upon your network topology, you may want to position intrusion detection systems at one or more places. It also depends upon what type of intrusion activities you want to detect: internal, external or both. For example, if you want to detect only external intrusion activities, and you have only one router connecting to the Internet, the best place for an intrusion detection system may be just inside the router or a firewall. If you have multiple paths to the Internet, you may want to place one IDS box at every entry point. However if you want to detect internal threats as well, you may want to place a box in every network segment. In many cases you don't need to have intrusion detection activity in all network segments and you may want to limit it only to sensitive network areas. Note that more intrusion detection systems mean more work and more maintenance costs. Your decision really depends upon your security policy, which defines what you really want to protect from hackers. Figure 1.4 shows typical locations where you can place an intrusion detection system [17].

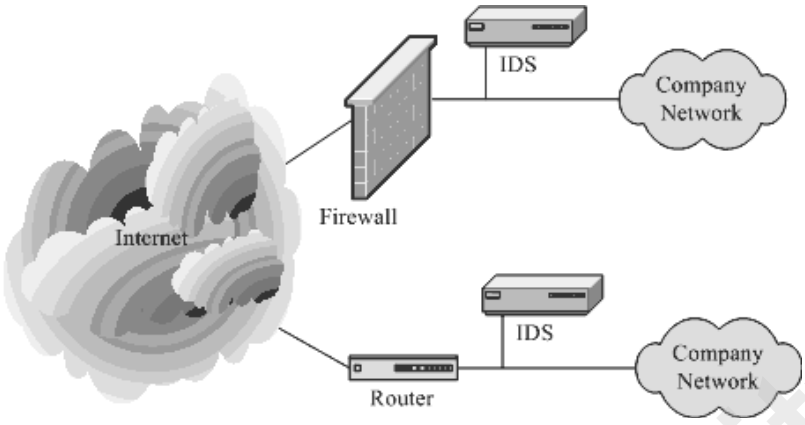


Figure 1.4: Typical locations for an intrusion detection system [17]

1.3 Introduction to Deep Packet Inspection

The proliferation of public and private networks and the increasing sophistication of network protocols and applications have driven a rapid escalation in the number and severity of attacks against computing systems, as shown in Figure 1.5.

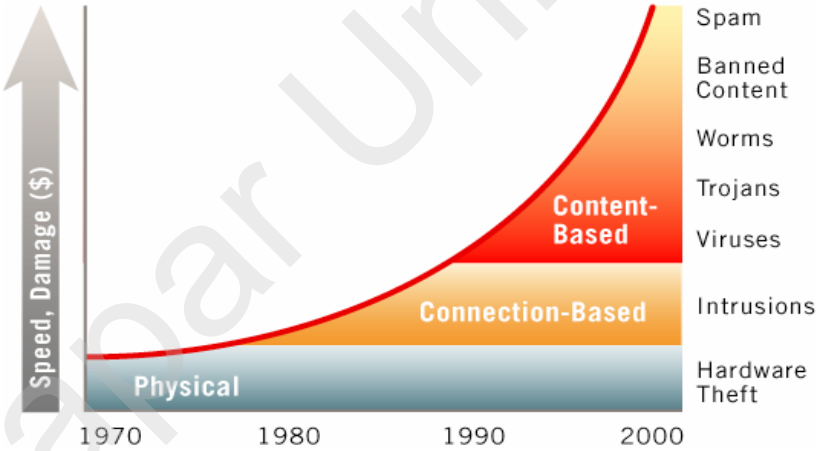


Figure 1.5: Severity of Attacks [9]

Early network protocols, such as Telnet, RPC and FTP, were relatively simple and required action by a dedicated hacker with a sustained connection to a remote system to launch an attack. The first incidences of this kind were identified by military organizations as intrusions to obtain classified information. The response to these types of attacks was the development of connection-oriented security systems, called Stateful Inspection firewalls, which control access to computing resources on a network by selectively allowing or denying the establishment of remote connections based primarily on the identity of the sender and receiver.

In the last ten years, applications have become much more complex, and protocols are used to carry much richer content. These changes have been exploited by attackers to develop more effective, content-based threats that circumvent connection-oriented security and that also have the ability to reproduce and spread automatically. Content-based threats are able to bypass connection-oriented Stateful-Inspection firewalls because they are typically delivered via connections that are inherently "trusted". Content-based threats include viruses, Trojans, worms, banned content and spam, and are readily propagated through email, web pages and other real-time communications applications.

1.3.1 Deep Packet Inspection

As mentioned above, most firewalls utilize Stateful Inspection technology, which works at the network layer to track each connection traversing all interfaces of the firewall to ensure validity. Decisions on whether or not to accept the packets are based upon policies established by the network administrator related to which senders are allowed to reach designated computing systems on their internal network, and which protocols they can use to exchange information. While this filtering is useful, it is not adequate to determine the difference between, say, a valid email message or a message infected with a virus, because Stateful Inspection does not check the actual contents of the packets to distinguish malicious content from valid content.

As shown in Figure 1.6, Stateful Inspection examines only the "headers" of data packets, which contain information such as the sender's and receivers' addresses and the type of protocol and data contained in the packet "payload". However, much as one might try to determine the value of a postal letter based only on the addresses on the outside of the envelope, the contents of the packet payloads themselves are not examined. As a result, Stateful Inspection technology cannot tell the difference between valid and harmful data if it originates from an otherwise "trusted" source such as an ISP's email server or any public Web site. Stateful Inspection is therefore effective only for preventing simple intrusions and other connection-based attacks.

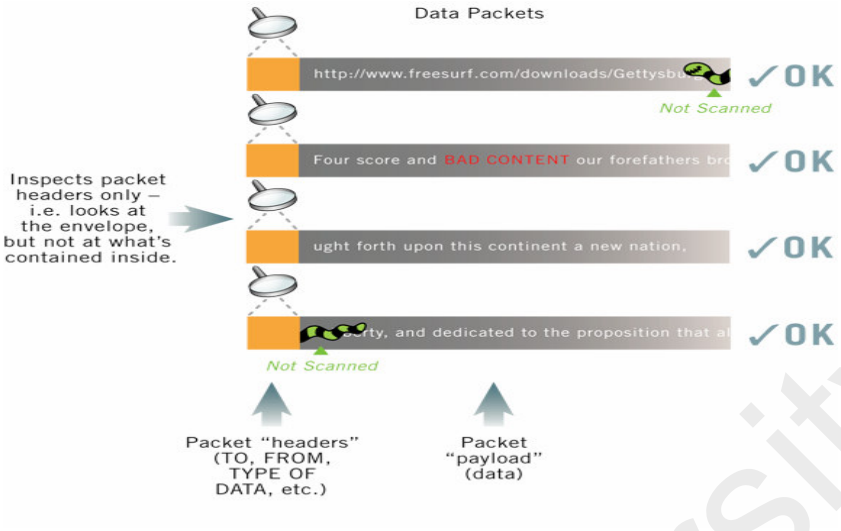
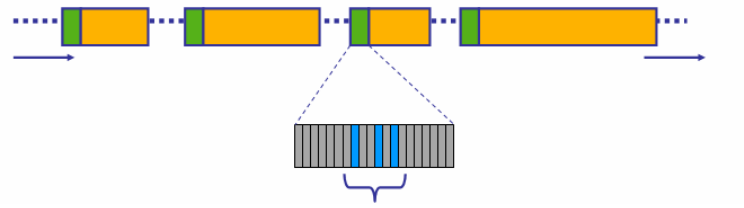


Figure 1.6: Stateful Inspection Firewall [9]

To address the limitations of Stateful Inspection, a technology known as Deep Packet Inspection (DPI) was developed. DPI goes further than Stateful Inspection to examine the payloads, or contents contained in packets in addition to the headers. As long as an attack can be contained in just a few packets, DPI can be effective in detecting and ultimately preventing the attack. As such, DPI technology is effective against buffer overflow attacks, denial of service (DoS) attacks, sophisticated intrusions, and a small percentage of worms that fit within a single packet.

1.3.2 Shallow vs. Deep Packet Inspection

The standard packet inspection process is a kind of shallow packet inspection that extracts basic protocol information such as IP addresses (source, destination) and other low-level connection states. This information typically resides in the packet header itself and consequently reveals the principal communication intent.



Headers showing the intent of Communication

Figure 1.7: Shallow Packet Inspection – data from packet headers [2]

The inspection level in the shallow inspection process is insufficient to reach any application-related deductions. For example, if a packet is the result of an application trying to set up additional connections for its core operation, an examination of the source or destination addresses as they appear within the packet header itself will not reveal any useful information regarding the connections to be used in the future, as requested by the application. Furthermore, as in this example, it is very common that the necessary information is spread over several packet transactions; and once again, examination of the header information alone overlooks the complete transaction perspective. DPI, on the other hand, provides application awareness. This is achieved by analyzing the content in both the packet header and the payload over a series of packet transactions. Consequently, DPI provides the ability to analyze network usage and optimize network performance, thereby playing a crucial role in the equation between supply and demand faced by every network operator.

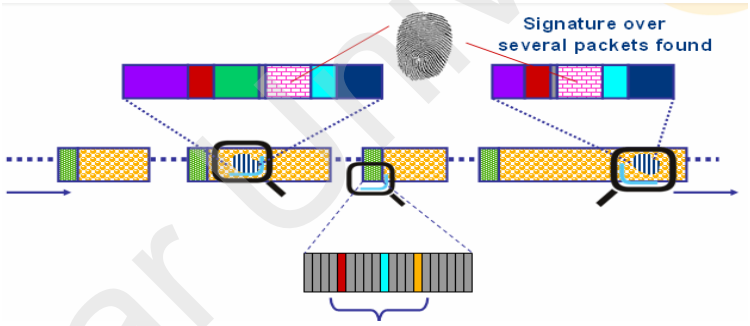


Figure 1.8: Deep Packet Inspection [2]

Chapter-2
Literature Survey

2.1 Classifications of Intrusion Detection System

Intrusion Detection Systems can be categorized based on several criteria. These are:

- Deployment based classification
- Scope based classification
- Techniques based classification

2.1.1 Deployment based classification

On the basis of deployment IDS can be categorized broadly into flowing two types:

- **Inline IDS**

Inline IDSs are placed in the line of packet transfer. IDS checks all packets passing through it for intrusions and if detected, alerts the network administrator. In addition, it can block the intrusion from going any further. Typically, IIDS are placed at the perimeter of the network, behind the firewall or along with the firewall.

- **Sniffer IDS**

These are IDSs that snoop and examine packets within a network. Sniffer IDSs do not come in the way of packet transfer. Instead, they passively receive all the packets from the network and perform intrusion analysis on these packets. Typically, sniffers are placed in a location on the network where they can read all the packets flowing through the network [12].

2.1.2 Scope based classification

When considering the area being the source of data used for intrusion detection, another classification of intrusion detection systems can be used in terms of the type of the protected system[12]. These are:

- Host Based IDS
- Network Based IDS
- Honeypot IDS
- Hybrid IDS
- GrIDS

2.1.2.1 Host Based IDS

Host based IDS consists of software or AGENT components, which exist on a Server, Router, Switch or Network appliance [15]. The agent versions must report to a console or can be run together on the same Host. This is NOT the preferred method though. The ideal method is illustrated in Figure 2.1[26].

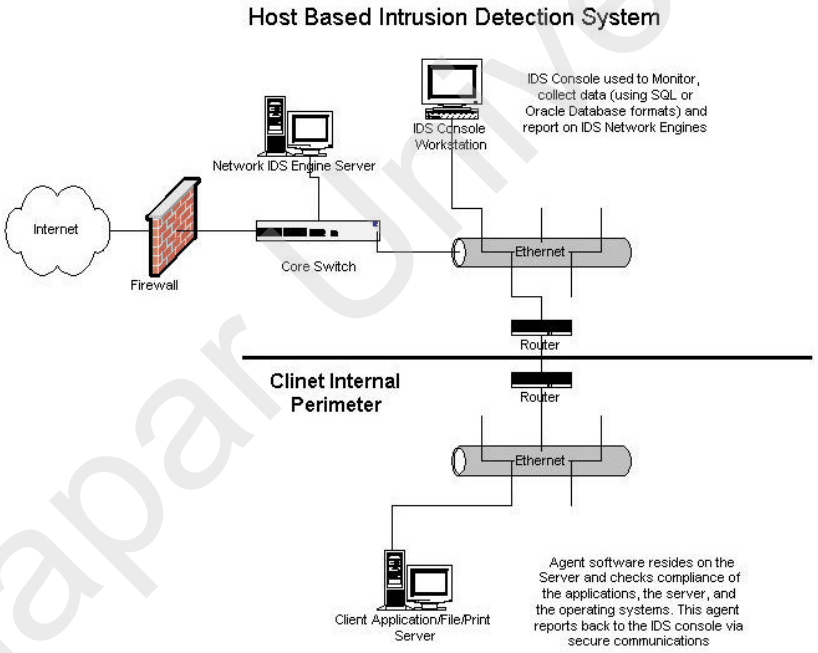


Figure 2.1: Host Based IDS[26]

The above model allows for remote monitoring, remote storage of event logs and the ability to PUSH agents to new or existing Hosts.

2.1.2.2 Network Based IDS

Network based IDS captures network traffic packets (TCP, UDP, IPX/SPX, etc.) and analyzes the content against a set of rules or signatures to determine if a possible event took place. False positives are common when an IDS system is not

configured or "tuned" to the environment traffic it is trying to analyze. Network Node is merely an extended model of the networked IDS systems adding segregated and dedicated IDS servers on each NODE of a network in order to capture all the networked traffic not visible to other IDS servers.

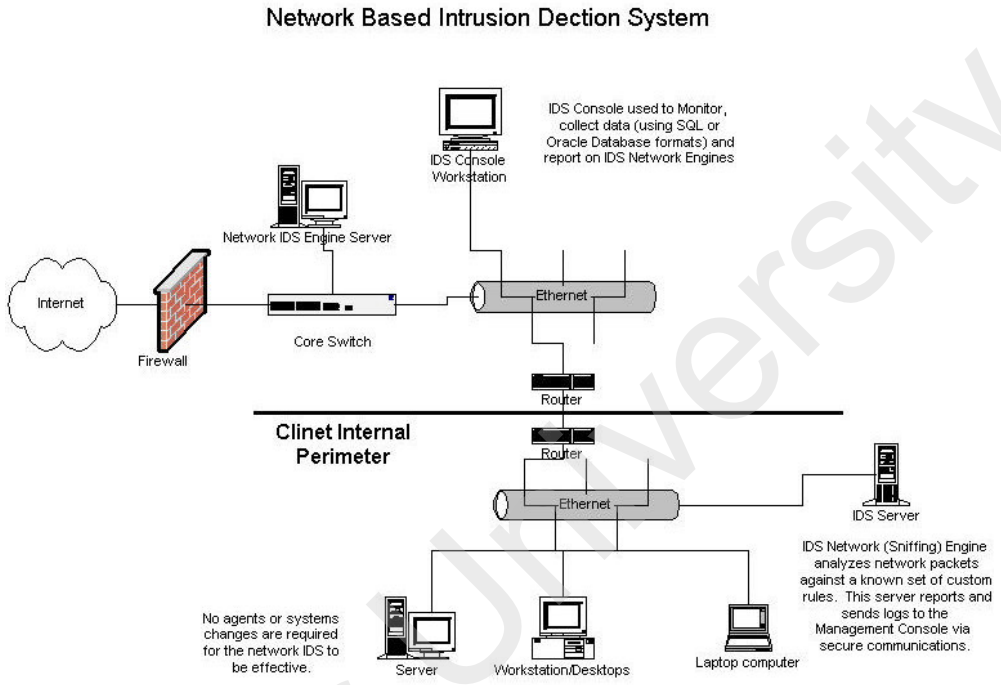


Figure 2.2: Network Based IDS [26]

2.1.2.3 Honeypot IDS

Honeypots are systems used to lure hackers by exposing known vulnerabilities deliberately. Once a hacker finds a honey pot, it is more likely that the hacker will stick around for some time. During this time hacker activities can be logged to find out hacker's actions and techniques. Once anyone knows these techniques, this information can be used later on to harden security on the actual servers. Honeypot Intrusion Detection Systems are utilized as a "Bait" and "Trap" methodology. They can emulate numerous operating systems, multiple operating systems, dedicated "services" (File Transfer Protocol (FTP), Hyper Text Transfer Protocol (HTTP), etc.) and most importantly, TRACK the 'finger print" of an intruder. This allows the tracing of intruders to the source or origin of the attack. Obviously there are inherent risks in using Honeypot technology, the allowing of a "Hacker" onto a computer is a dangerous but if placed correctly in and environment this is limited to the Honeypot Server and the fact that you have

"available" services or operating systems noticeable on your network can bring more "hackers" to snoop the network. Again, this is avoidable if placed in the right location and isolated from all other "Operational" systems.

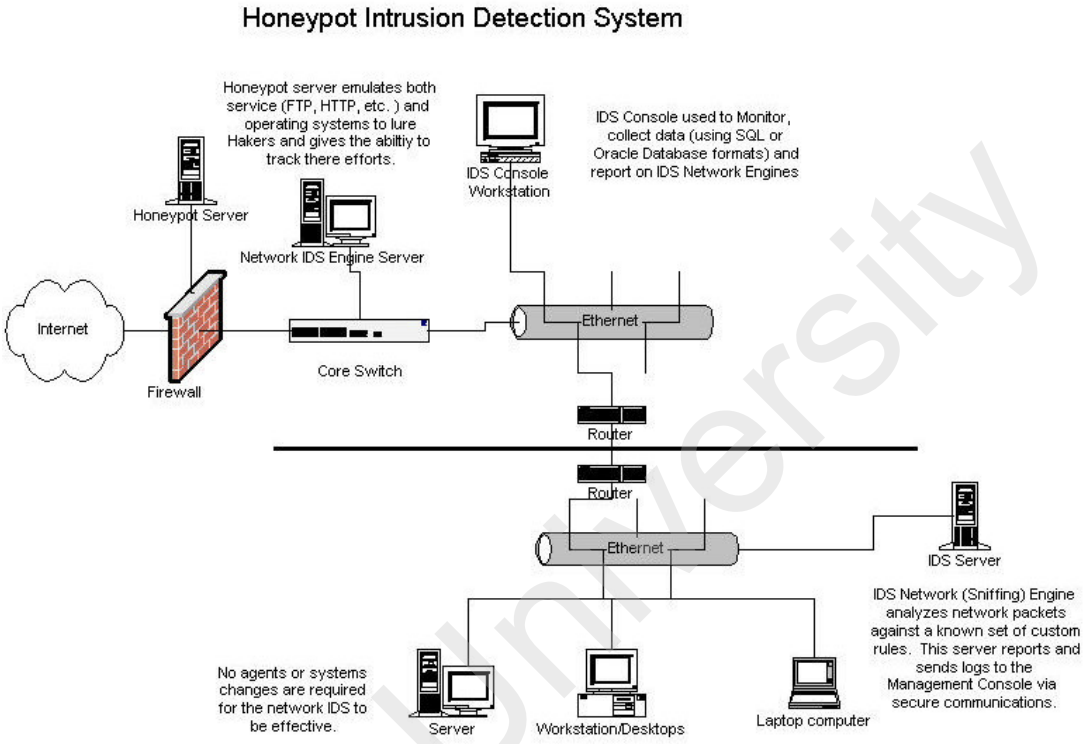


Figure 2.3: Honeypot IDS[26]

2.1.2.4 Hybrid IDS

Hybrid Intrusion Detection Systems are the "IDS of the Future". They exist in a limited form but are still on the cutting edge of technologies for IDS itself. They are based on a model which brings multiple agents of multiple types such as Simple Network Message Protocol (SNMP), Packet "sniffers", TCPdump (a Unix based network packet capture and filtering software), Cybercop Scanner, ISS Security Scanner, Snort IDS and other multi-vendor IDS systems all of which come into a Centralized Management Console and are analyzed by its centralized processors. Typically these product multi- vendor products are accumulated into a central Database Program or Engine such as SQL or Oracle. The purpose of this technology is to bring more flexibility, expandability and most important to cross-check anomalies against other systems to enhance there accuracy of the alerts and reduce "False Positives".

2.1.2.5 GrIDS

Graph Based Intrusion Detection System (GrIDS) is built to detect distributed attacks against large networks. It constructs activity graphs to represent connections between hosts (network traffic) in its domain. Every node in the graph represent a single host or a set of hosts (domain) and every edge in the graph represents the network traffic between nodes. GrIDS uses network sniffers and also host based ID systems as its data source. At this point it is said to be a combined approach of host based and network based IDS. Collected data will be the input of the graph engine, which is responsible for creating activity graphs [10].

2.1.3 Techniques based classification

Intrusions can be detected by various techniques. Most important of those are:

- **Anomaly detection**

Normal behavior patterns are useful in predicting both user and system behavior. Here, anomaly detectors construct profiles that represent normal usage and then use current behavior data to detect a possible mismatch between profiles and recognize possible attack attempts.

- **Signature detection**

Systems possessing information on abnormal, unsafe behavior (attack signature-based systems) are often used in real-time intrusion detection systems (because of their low computational complexity).

2.2 Techniques used to detect Intrusions

Intrusion detection systems must be capable of distinguishing between normal that is not security-critical and abnormal user activities, to discover malicious attempts in time. However translating user behaviors in a consistent security-related decision is often not that simple because many behavior patterns are unpredictable and unclear as shown in Figure 2.4. In order to classify actions various intrusion detection techniques are there [16]. These can broadly be classified into the following categories:

- Anomaly Based
- Signature Based

Figure 2.4: Behavior of the user in the system[16]

2.2.1 Anomaly detection--Normal behavior patterns

Normal behavior patterns are useful in predicting both user and system behavior. Here, anomaly detectors construct profiles that represent normal usage and then use current behavior data to detect a possible mismatch between profiles and recognize possible attack attempts [8].

In order to match event profiles, the system is required to produce initial user profiles to train the system with regard to legitimate user behaviors. There is a problem associated with profiling: when the system is allowed to "learn" on its own, experienced intruders (or users) can train the system to the point where previously intrusive behavior becomes normal behavior. An inappropriate profile will be able to detect all possible intrusive activities. Furthermore, there is an obvious need for profile updating and system training which is a difficult and time-consuming task. Given a set of normal behavior profiles, everything that does not match the stored profile is considered to be a suspicious action. Hence, these systems are characterized by very high detection efficiency (they are able to recognize many attacks that are new to the system), but their tendency to generate false alarms is generally a problem.

Advantages of this anomaly detection method are:

- Possibility of detection of novel attacks as intrusions;
- Anomalies are recognized without getting inside their causes and characteristics;
- Less dependence of IDSs on operating environment (as compared with attack signature-based systems);
- Ability to detect abuse of user privileges.

The biggest disadvantages of this method are:

- A substantial false alarm rate.

- User behaviors can vary with time, thereby requiring a constant update of the normal behavior profile database.
- The necessity of training the system for changing behavior makes a system immune to anomalies detected during the training phase.

2.2.2 Signature detection-- Misbehavior signatures

Systems possessing information on abnormal, unsafe behavior (attack signature-based systems) are often used in real-time intrusion detection systems.Signature is a methodical and systematic identification process must be employed. Similar to a lawful operation in which fingerprints are used to identify the involvement of individuals in specific incidents, signatures are used to identify applications and protocols. In their most broad sense, signatures are pattern recipes which are chosen for uniquely identifying an associated application (or protocol). When a new application or protocol is encountered, it is analyzed and an appropriate signature is developed and added to a database typically referred to as a signature library.

Methods of Signature Analysis

There are several possible methods of analysis used to identify and classify traffic. These range from analysis by

- Port
- String match
- Numerical properties
- Behavior and Heuristics [2]

- **Analysis by Port**

Analysis by port is probably the easiest and most well known form of signature analysis. The reasoning is the simple fact that many applications use either default ports or some chosen ports in a specific manner. However, since it is very easy to detect application activity by port, this is in fact a weakness, particularly because many current applications disguise themselves as other applications. The most common example is the Port 80, where many applications seem as pure HTTP traffic. As noted above, some applications select random ports instead of using fixed default ports. In this case, there is often some pattern involved in the port selection process – for example, the first port may be random, but the next will be

the subsequent one, and so forth. However in some cases the port selection process may be completely random.

For all these reasons, it is often not feasible to use analysis by port as the only tool for identifying applications, but rather as a form of analysis to be used together with other tools.

- **Analysis by String Match**

Analysis by string match involves the search for a sequence of textual characters or numeric values within the contents of the packet. Furthermore, string matches may consist of several strings distributed within a packet or several packets. For example, many applications still declare their names within the protocol itself, as in Kazaa, where the string "Kazaa" can be found in the User-Agent field with a typical HTTP GET request. If analysis is performed by port analysis alone, then port 80 may indicate HTTP traffic and the GET request will further corroborate this observation. However, since the User-Agent field information is missing, this analysis will result in inaccurate classification i.e., HTTP and not Kazaa.[2]

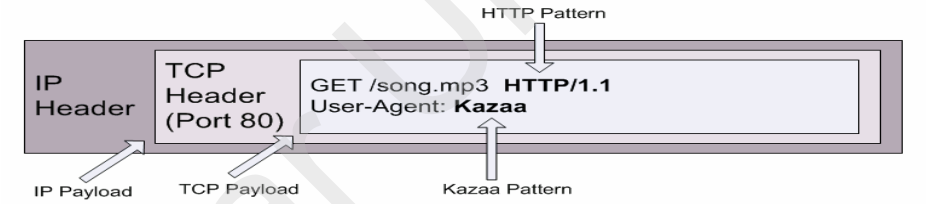


Figure 2.5: Kazaa string match analysis[2]

This example emphasizes once again that several analysis tools are required for assuring proper classification. Another example below presents an attack signature taken from the Snort program that detects ping ICMP packets larger than 800 bytes, incoming from an external network and associated with any port:

alert icmp $EXTERNAL_NET any -> $HOME_NET any (msg:"MISC large ICMP"; dsize: >800; reference:arachnids,246; classtype:bad-unknown; sid:499;)

- **Analysis by Numerical Properties**

Analysis by numerical properties involves the investigation of arithmetic and numerical characteristics within a packet, and of a packet or several packets. Some examples of properties analyzed include payload length, the number of packets

sent in response to a specific transaction, and the numerical offset of some fixed string or byte value within a packet.

- **Analysis by Behavior and Heuristics**

Behavioral analysis refers to the way a protocol acts and operates. Heuristic analysis typically boils down to the extraction of statistical parameters of examined packet transactions. Often, behavioral and heuristic analysis are combined to provide improved assessment capabilities. For example, actions leading to other actions can clearly indicate a behavioral pattern which can be traced, as in the case where an active UDP connection eventually transforms into a TCP.

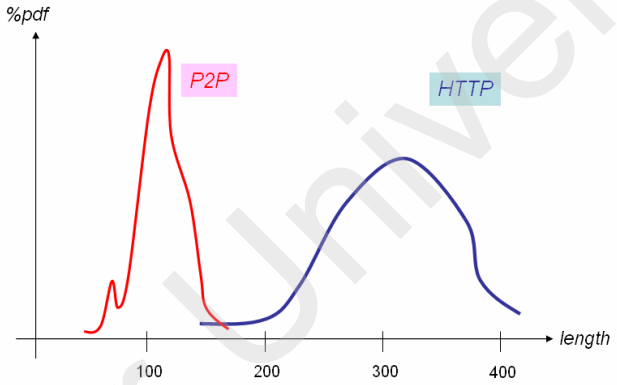


Figure 2.6: HTTP vs. P2P [2]

Another example of behavior and heuristic analysis is shown in Figure 2.6, which compares HTTP and a typical P2P file sharing application. If the packet length histogram (PDF) alone is examined while ignoring the file download or upload transaction itself (which tends to use large packet lengths), it becomes apparent that while pure HTTP packets tend to concentrate around a few hundred bytes in length, P2P control layer information tends to use shorter packet lengths. In this way, by examining some short-term statistics, it is possible to conclude whether a port 80 connection carries pure HTTP traffic or other P2P-related traffic.

Advantages of Signature Based IDS

The signature detection methods have the following advantages:

- Very low false alarm rate

- Simple algorithms
- Easy creation of attack signature databases
- Easy implementation and typically minimal system resource usage.

Disadvantages of Signature Based IDS

Some disadvantages:

- Difficulties in updating information on new types of attacks (when maintaining the attack signature database updated as appropriate).
- They are inherently unable to detect unknown, novel attacks. A continuous update of the attack signature database for correlation is a must.
- Maintenance of an IDS is necessarily connected with analyzing and patching of security holes, which is a time-consuming process.
- The attack knowledge is operating environment–dependent, so misbehavior signature-based intrusion detection systems must be configured in strict compliance with the operating system (version, platform, applications used etc.)
- They seemed to have difficulty handling internal attacks. Typically, abuse of legitimate user privileges is not sensed by the system as a malicious activity (because of the lack of information on user privileges and attack signature structure)**.**

2.2.3 Parameter Pattern Matching—Unpredictable behaviour

The third method of intrusion detection is subtler than the two mentioned earlier. It reasons on the fact, that system administrators monitor various systems and network attributes (not necessarily targeting security issues). As a rule, information obtained in this way has a constant specific environment. This method involves the use of day-to-day operational experience of the administrators as the basis for detecting anomalies. It can be considered as a special case of Normal Profile Methods. The difference lies in the fact that a profile here is part of the human knowledge. This is a very powerful technique, since it allows intrusions based on unknown type attacks. The system operator can detect subtle changes that are not obvious to the operator himself. Its inherent disadvantage is connected with the fact that humans can process and hence understand only a limited portion of information at a time, what means that certain attacks may pass undetected [16].

2.3 Bro : A Network based Intrusion Detection System

Due to the enormous increase in the number of attacks seen on the Internet, *Network Intrusion Detection Systems* (NIDSs) are an important part of many security policies. In contrast to a *host-based* intrusion detection system**,** a NIDS usually taps the network at some central point, monitoring all traffic to detect malicious activity. Traditionally, two approaches to network intrusion detection are differentiated: a system using *anomaly-detection* relies on a definition of *normal* network activity and alerts if traffic deviates from this profile. Deviation is typically measured using statistics. On the other hand, *misuse-detection* systems use a library of specific attack characteristics and generate an alert whenever they locate one of them in the monitored traffic. The draw-back of misuse-detection is the difficulty of recognizing previously unknown attacks. While anomaly-based NIDSs have the potential for this, they often suffer from a high number of false-alarms. The most common form of misuse detection is signature matching: the NIDS identifies an attack by recognizing a specific byte sequence within the network packets. Most commercial systems follow this approach as does the well-known open source software Snort . Taking a closer look at open source NIDSs, we notice that Snort is the only one widely deployed in real networks. It appears that most people are not aware of alternatives. This is interesting for three reasons. First, in other areas of security there usually is a variety of open source software available. Second, Snort shows some technical limitations that indicate that it is not always an optimal solution. Finally, there is at least one other very powerful open source system that is largely unknown: Bro [3].

2.3.1 Brief Description about Bro

Bro originated as a research system. It is designed and developed by Vern Paxson of *ICSI's Center for Internet Research (ICIR), Berkeley*. He started the project in 1995 at *Lawrence Berkeley National Laboratory (LBL)*, and Bro is under active development since then. It was first published in 1998. Major extensions of Bro are contributed by people at the *University of California, Berkeley* and *Princeton University, New Jersey*. Among other locations, Bro is operationally deployed at the *University of California, Berkeley* and at LBL.

Bro's fundamental design goal is to separate mechanism and policy. While Bro implements very sophisticated state management and protocol analysis, it is by itself *policy neutral*. The network activity is abstracted into *events* which are passed from Bro's core to an upper layer, the *policy layer*. On this policy layer, the administrator defines environment specific constraints by writing custom scripts in a powerful scripting language. As a consequence of this design, Bro is neither fundamentally anomaly-based nor misuse-based. Instead it supports both approaches and, in fact, the default policy scripts shipped with Bro implement examples of anomaly- as well as misuse-detection. Its integrated signature matcher provides a superset of Snort's capabilities and may even use Snort's signatures by means of a converter.

2.3.2 Architecture of Bro

Bro is conceptually divided into an "event engine" that reduces a stream of (filtered) packets to a stream of higher-level network events, and an interpreter for a specialized

language that is used to express a site's security policy. More generally, the system is structured in layers, as shown in Figure 2.7. The lower-most layers process the greatest volume of data, and hence must limit the work performed to a minimum. As we go higher up through the layers, the data stream diminishes, allowing for more processing per data item.

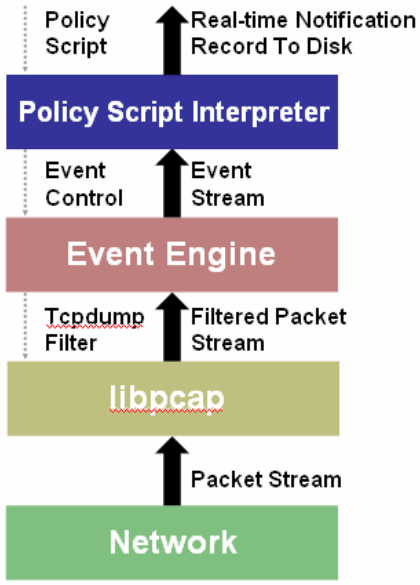


Figure 2.7: Structure of the Bro system [24]

A) libpcap

Just above the network layer is libpcap [22], the packet capture library used by tcpdump. There are several advantages of having libpcap in Bro:

1) It isolates Bro from details of the network link technology (Ethernet, FDDI, SLIP, etc.).
2) It greatly aids in porting Bro to different Unix variants (which also makes it easier to upgrade to faster hardware as it becomes available).
3) It means that Bro can also operate on tcpdump save files, making off-line development and analysis easy.
4) If the host operating system provides a sufficiently powerful kernel packet filter, such as BPF, then libpcap downloads the filter used to reduce the traffic into the kernel.

The key to packet filtering is, of course, judicious selection of which packets to keep and which to discard. Tcpdump can also use some filters to discard those packets which are not required and accept only useful ones. Like in the following example: *tcp port ftp or tcp port telnet*

That is, the filter accepts any TCP packets with a source or destination port of 21 (FTP) or 23 (Telnet).

B) Event engine

The resulting filtered packet stream is then handed up to the next layer, the Bro "event engine." This layer first performs several integrity checks to assure that the packet headers are well-formed. If these checks fail, then Bro generates an event indicating the problem and discards the packet. If the checks succeed, then the event engine looks up the connection state associated with the tuple of the two IP addresses and the two TCP or UDP port numbers, creating new state if none already exists. It then dispatches the packet to a handler for the corresponding connection. The following are the major tasks performed by an event engine:

• Performs integrity check on packet headers

• Reassembles IP fragments to analyze complete IP datagrams

• Checks connection state, creating new state if none already exists

• After processing each event, the event engine checks for new events kept in a FIFO queue

• If so, it processes the new events

• Checks for expired timers

C) Policy script interpreter

After the event engine has finished processing a packet, it then checks whether the processing generated any events. (These are kept on a FIFO queue.) If so, it processes

each event until the queue is empty, as described below. It also checks whether any timer events have expired, and if so processes them, too. A key facet of Bro's design is the clear distinction between the generation of events versus what to do in response

to the events. This structure reflects the separation between mechanism and policy. The "policy script interpreter" executes scripts written in the specialized Bro language. These scripts specify event handlers, which are essentially identical to Bro functions except that they do not return a value. For each event passed to the interpreter, it retrieves the semi-compiled code for the corresponding handler, binds the values of the events to the arguments of the handler, and interprets the code. This code in turn can execute arbitrary Bro scripting commands, including generating new events, logging real-time notifications (using the Unix *syslog* function), recording data to disk, or modifying internal state for access by subsequently invoked event handlers (or by the event engine itself). Finally, along with separating mechanism from policy, Bro's emphasis on asynchronous events as the link between the event engine and the policy script interpreter buys a great deal in terms of extensibility. Adding new functionality to Bro generally consists of adding a new protocol analyzer to the event engine and then writing new event handlers for the events generated by the analyzer. Neither the analyzer nor the event handlers tend to have much overlap with existing functionality, so for the most part the subtle interactions between loosely coupled modules can be avoided. The major tasks of policy script interpreter are:

• For each event passed to the interpreter it retrieves compiled code for the corresponding handler and binds values of the events to the arguments of the handler and interprets the code

• That code can in turn generate new events, log notifications, record data to disk, or modify the current state [25].

Chapter 3
Problem Statement

3.1 Problem Statement

Today's network is very complex and the whole world is focusing on ease of use and functionality. This is making us more insecure. For hackers, these 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. So there is a huge need of detecting the threats and intrusion. For this purpose number of solutions is there, IDS is one of them. BRO is the most effective NIDS which can be used to detect these threats. However no IDS can detect all the intrusions. So we need a combination of various techniques.

We will work on honing a number of techniques addressing intrusions involving a number of popular classes of network traffic. There are a number of well-known techniques for detecting network intrusions. Some of these are:

- Signatures or pattern matching
- Content analysis and parsing
- Statistical analysis
- Anomaly detection
- Bayesian methods

Each of these techniques has their relative pros and cons. Some are easy to use and quick to implement, but lead to a high number of false-positives. Others are hard to understand and are complex, but at the same time may be very effective in detecting desired flows.

The objective is to analyze these techniques and develop patterns (not in the regular expression sense but in the software patterns one) that direct us in writing effective intrusion detection modules for a variety of network traffic classes. The types of traffic which are of interest are:

- Web traffic, usually sent over HTTP [11] protocol.
- E-mail traffic, using one of the well known e-mail protocols like SMTP, POP3 and IMAP [13].
- Webmail traffic. This is placed in a separate category because this combines properties of both web and e-mail traffic.

These techniques will be implemented by developing Policy Scripts using Bro IDS. Both Bro analyzers and scripts will be used to achieve the goal. Bro language can be a bit limiting while trying to do advanced programming tasks.

Chapter-4

Implementation Details and Experimental Results

In this part the policy scripts of Bro to detect various network intrusions are discussed. Various kind of traffic is captured by using Bro and analyzed offline. Some of the scripts are experimented on the live traffic also.

4.1 Implementation Setup

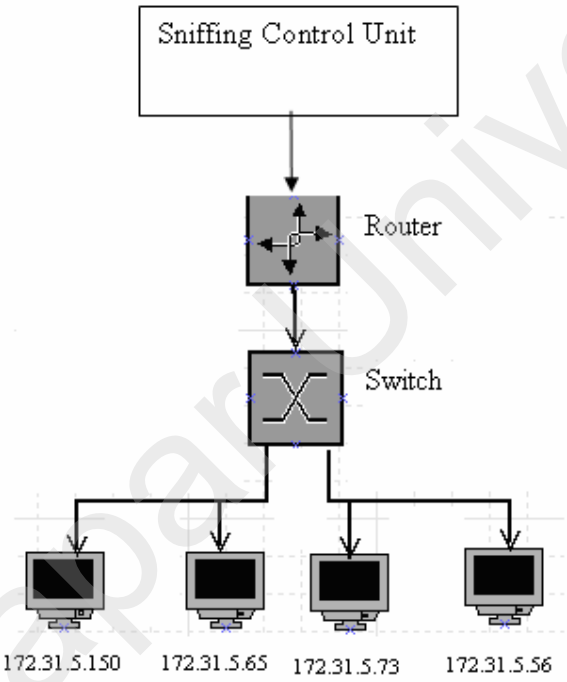


Figure 4.1: Implementation setup Diagram

4.2 Basic Steps performed while implementing Policy Scripts

1. First the traffic is captured by using wireshark [28] (earlier Ethreal) or by using libpcap feature inbuilt in Bro. This will be captured in a binary file with extension *.tcpdump* or *.out*, for example *trace.out.*

2. Second, the policy script is written using Bro Scripting language. This file is having an extension *.bro* like *s_http.bro* and is placed in */usr/local/bro/policy* or */user/local/bro/site* directory.

3. Third, this script is run against the captured trace file to detect the required intrusion by using the following syntax:

bro –r tracefile scriptfile

4. If the traffic is live instead of captured one then also bro can analyze it by using *–i* used for interface instead of *–r* for read mode like:

bro –i scriptfile

4.3 Milestones covered and Experimental Results

The following are various tasks and milestones completed and their results. One of the traffic files were taken from the Bro Workshop 2007 [4]. This was modified using wireshark and later used in some of the experiments. The name of this traffic file is mail.trace.

4.3.1 Reporting all the HTTP URLs in the traffic

In this milestone all the URLs which are visited by a particular machine are reported by the Bro.

Script: s_http-header.bro

```
# Script to report all the HTTP URLs in the traffic.
@load weird
@load alarm
@load http
global path: string;
redef ignore_checksums = T;
event http_request(c: connection, method: string, original_URI: string,
unescaped_URI: string, version: string)
    {
      path = original_URI;
    }
```

```
event http_header(c: connection, is_orig: bool, name: string, value: string)
    {
        if(name == "HOST" )
        {
            print fmt("URL: %s%s",value,path );
        }
    }
```

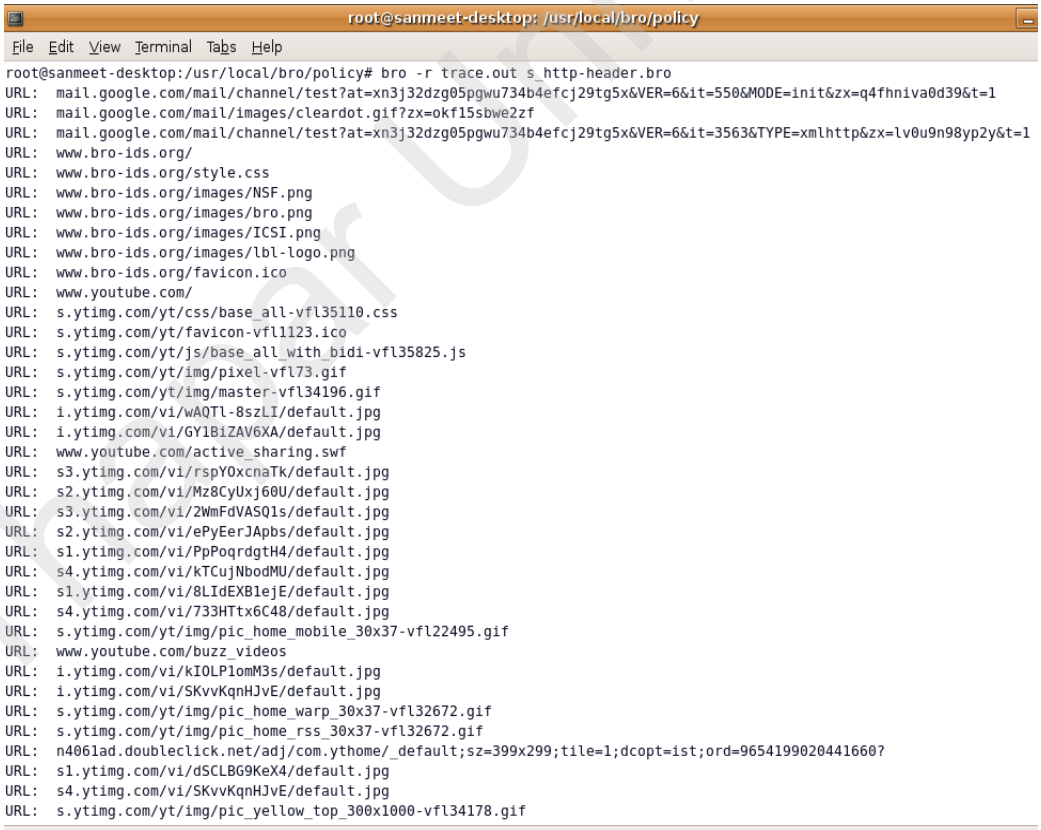
Trace File: trace.out

This is the captured traffic file against which s_ http-header.bro policy script will run.

The following is the command to see the results:

bro -r trace.out s_http-header.bro

Result:



Snapshot 4.1: Reporting HTTP URLs visited by a host.

4.3.2 Reporting all the connections which are accessing www.youtube.com using HTTP

In this milestone all the connections which are accessing www.youtube.com using HTTP are reported by Bro.

Script: s_http-header1.bro

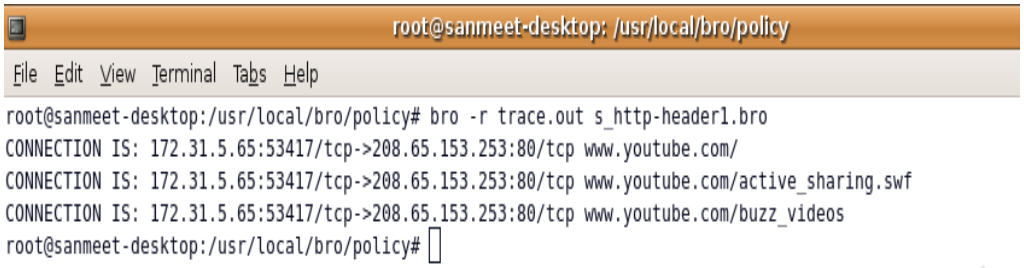
```
# Bro script to report all the connections that are accessing www.youtube.com using HTTP
@load weird
@load alarm
@load http
global path: string;
redef ignore_checksums = T;
event http_request(c: connection, method: string, original_URI: string,
 unescaped_URI: string, version: string)
      {
       path = original_URI;
      }
event http_header(c: connection, is_orig: bool, name: string, value: string)
      {
            if(name == "HOST" )
            {
              local v = value;
              if("www.youtube.com" in v )
                {
print fmt("CONNECTION IS: %s:%s->%s:%s%s%s",
c$id$orig_h,c$id$orig_p,c$id$resp_h,c$id$resp_p,v,path);
                  }
            }
      }
```

Trace file: trace.out

The above is the captured traffic file against which s_ http-header1.bro policy script will run. The following is the command to see the results:

bro -r trace.out s_http-header1.bro

Result:



Snapshot 4.2: Reporting all the connections which are accessing www.youtube.com using HTTP

4.3.3 Reporting all the connections that includes emails directed to a particular email server

In this milestone all the connections that have a particular text like "@beta.banana.edu" in them are reported by Bro. The intention is to find out all emails that might be addressed to this account. SMTP [13] protocol support is required in this task

Script : s_smtp.bro

Bro script that reports all the connections that have the text "@beta.banana.edu" in them.

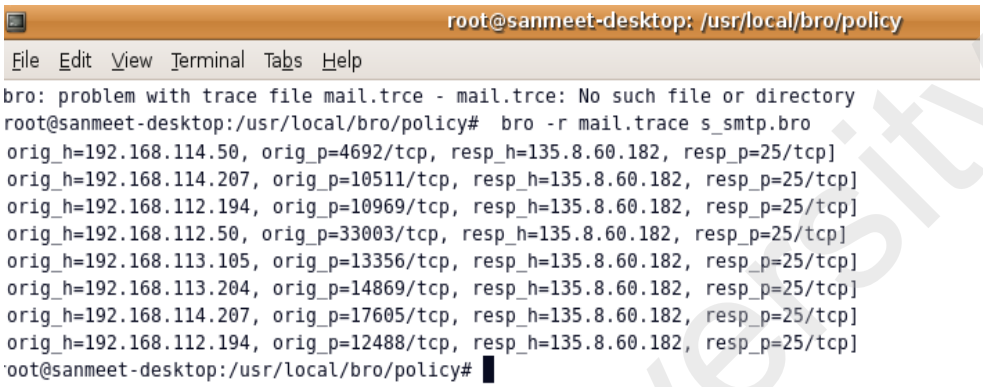
```
@load weird
@load alarm
@load smtp
event smtp_request(c: connection, is_orig: bool, command: string, arg: string)
    {
    local id = c$id;
    if ( command == "RCPT" )
      {
     if("@beta.banana.edu" in arg)
    {
        print(id);
        }
    }
    }
```

Trace File: mail.trace [Bro workshop 2007]

The above is the captured mail traffic file against which s_ smtp.bro policy script will be experimented . The following is the command to see the results:

bro -r mail.trace s_smtp.bro

Result:



Snapshot 4.3: Reporting all the connections that have text "@beta.banana.edu"

4.3.4 Reporting all the connections that have a particular text like "@beta.banana.edu" or "@finch.eyrie.af.mil" in them

In this milestone all the connections that have a particular text like "@beta.banana.edu" or "@finch.eyrie.af.mil" in them are reported by Bro. The intention is to find out all emails that might be addressed to any of these accounts. SMTP protocol support is required in this task

Script: s_smtp1.bro

```
# Bro Script that Reports all connections that have text like "@beta.banana.edu"
or "@finch.eyrie.af.mil" in them
@load weird
@load alarm
@load smtp
event smtp_request(c: connection, is_orig: bool, command: string, arg: string)
{
        local id = c$id;
        if ( command == "RCPT" )
```

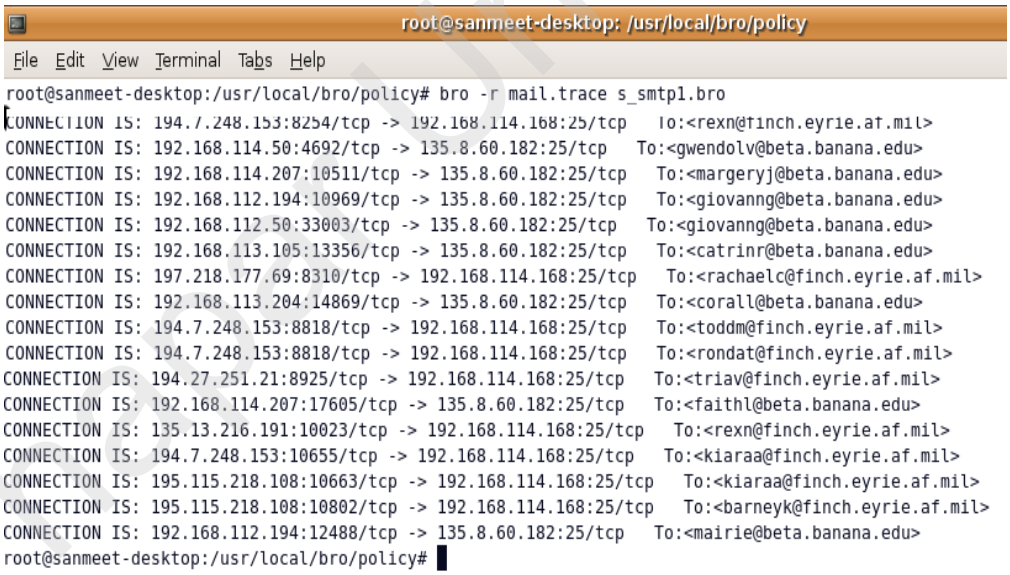
```
{
if("@beta.banana.edu" in arg || "@finch.eyrie.af.mil" in arg)
  {
  print fmt("CONNECTION IS: %s:%s -> %s:%s
  %s",c$id$orig_h,c$id$orig_p,c$id$resp_h,c$id$resp_p,arg);
  }
}
```

Trace File: mail.trace [Bro workshop 2007]

The above is the captured mail traffic file against which s_ smtp.bro policy script will be experimented . The following is the command to see the results:

bro -r mail.trace s_smtp1.bro

Results:



Snapshot 4.4: Reporting connections that have “@beta.banana.edu" or "@finch.eyrie.af.mil" in them

4.3.5 Detect if somebody is trying to access a particular website like "pic.geocities.com" using HTTP, log all further connection attempts by that host

In this milestone we have captured HTTP and other network traffic from multiple clients. Once it is detected that somebody is trying to access a particular website like "pic.geocities.com" using HTTP, all further connection attempts by that host are logged.

Script : s_exerercise5.bro

```
# Bro Script that log all connections that attempts to access a particular website like pic.geocities.com
@load weird
@load alarm
@load http
global path: string;
global cid: set[addr] &persistent;
redef ignore_checksums = T;
event http_request(c: connection, method: string, original_URI: string,
                   unescaped_URI: string, version: string)
      {
            path = original_URI;
      }
event http_header(c: connection, is_orig: bool, name: string, value: string)
      {
           if(name == "HOST" )
            {
             local v = value;
             if("pic.geocities.com" in v )
              {
                add cid[c$id$orig_h];
                print fmt("CONNECTION IS:
                %s:%s->%s:%s%s%s",cid,c$id$orig_p,c$id$resp_h,c$id$resp_p,v,path);
```

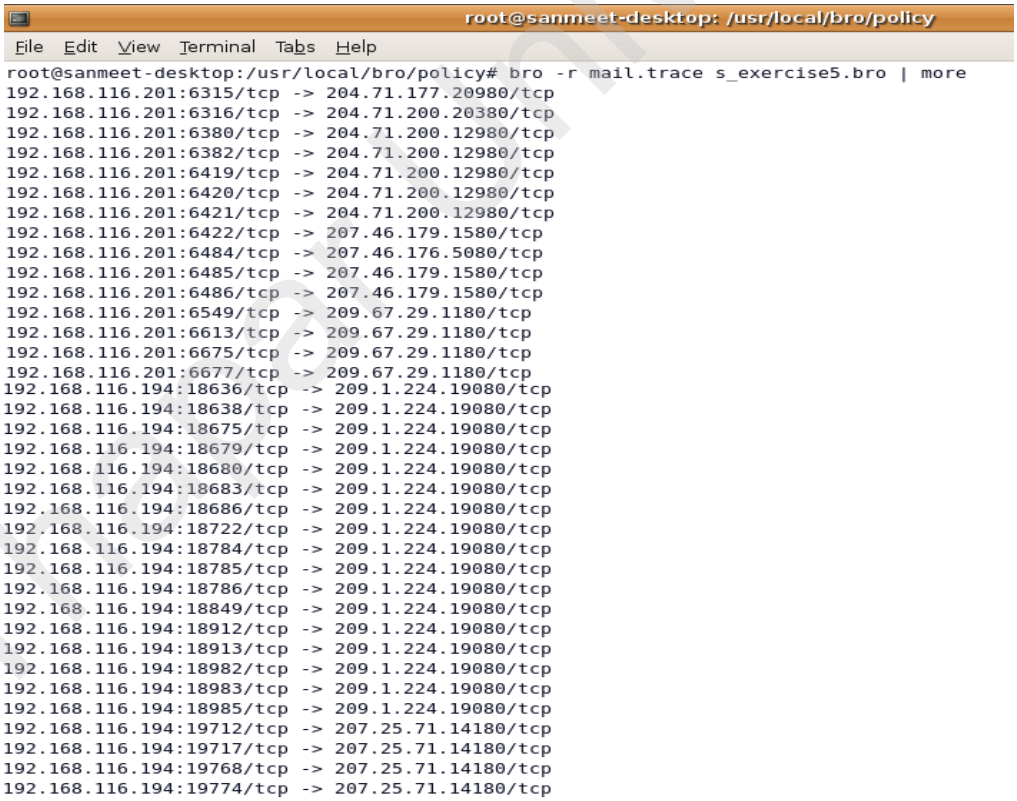
```
            }
          }
       }
event connection_established(c: connection)
{
     if (c$id$orig_h in cid)
     print fmt("%s:%s->
%s%s",c$id$orig_h,c$id$orig_p,c$id$resp_h,c$id$resp_p);
}
```

Trace File : mail.trace

The above is the captured traffic file against which s_ excercise5.bro policy script will be experimented. The following is the command to see the results:

bro -r mail.trace s_excercise5.bro

Results:



Snapshot 4.5 Logging all connections that attempts to access pic.geocities.com

4.3.6: Logging connections those attempt to access pic.geocities.com in a file instead of stdout

In this task we have captured HTTP and other network traffic from multiple clients. Once it is detected that somebody is trying to access pic.geocities.com using HTTP, all further connection attempts by that host are logged and written to a file, instead of stdout.

Script : s_exercise6.bro

```
@load weird
@load alarm
@load http
global path: string;
global cid: set[addr] &persistent;
global shanz_log = open_log_file("http") &redef;
redef ignore_checksums = T;
event http_request(c: connection, method: string, original_URI: string,
                   unescaped_URI: string, version: string)
      {
          path = original_URI;
       }
event http_header(c: connection, is_orig: bool, name: string, value: string)
      {
           if(name == "HOST" )
           {
             local v = value;
             if("pic.geocities.com" in v )
              {
                 add cid[c$id$orig_h];
              }
           }
      }
event connection_established(c: connection)
{
```

```
    if (c$id$orig_h in cid)
    print shanz_log, fmt
    ("%s:%s -> %s%s",c$id$orig_h,c$id$orig_p,c$id$resp_h,c$id$resp_p);
}
```

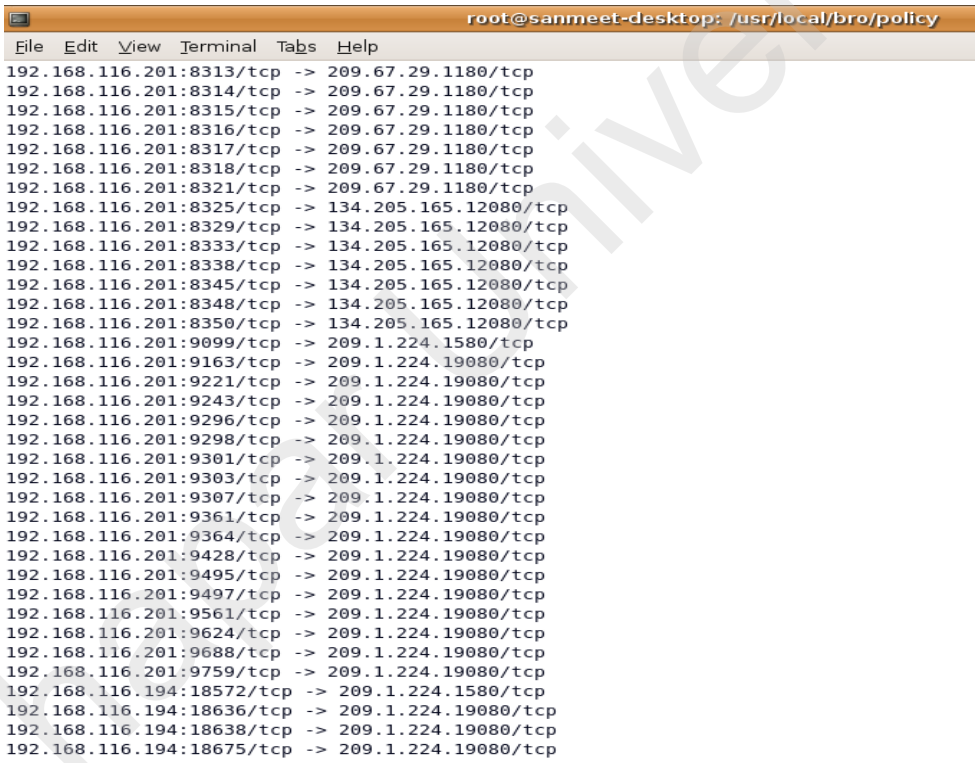
Trace File : mail.trace

The above is the captured traffic file against which s_ excercise6.bro policy script will be experimented. The following are the commands to see the results:

1*) bro -r mail.trace s_exercise6.bro*

2) less http.log

Snapshot of Result:



Snapshot 4.6: Logging connections those attempt to access pic.geocities.com in a file instead of stdout

4.3.7 Detecting all GTalk traffic in a captured file using a script

In this task Gtalk traffic is detected out of the captured traffic. There is no event for this. Here signatures are used. We have detected this with a captured traffic in some trace File.

Signature File : s_gtalk.sig

```
signature gtalk-1
{
ip-proto == tcp
dst-ip == 72.14.253.125
dst-port == 5222
event "GMAIL TRAFFIC"
 }
```

Signature File : s_gtalk1.sig

```
signature gtalk-2
{
ip-proto == tcp
src-port == 5222
event "GTALK Response TRAFFIC"
}
```

Script File : s_gtalkcap.bro

```
# Bro File to detect Gtalk Traffic Captured passively in some Trace File
global f: file =open("s1.trace");
redef signature_files += "s_gtalk";
redef signature_files += "s_gtalk1";
redef ignore_checksums = T;
event signature_match(state:signature_state, msg:string, data:string)
{
 local id=state$id;
 if(id=="gtalk-1")
{
 print(msg);
 }
if(id=="gtalk-2")
{
print(msg);
```

```
}
}
event connection_established(c:connection)
{
print(c$id);
set_contents_file(c$id,CONTENTS_ORIG,f);
}
event bro_done()
{
close(f);
}
```

Trace File: gtalk1.trace

The above is the captured traffic file against which s_gtalkcap.bro policy script will be experimented. The full packets can't be captured by this approach. Only contents of headers will be captured. Those contents are stored in a file called s1.trace.The following are the commands to see the results:

bro -r gtalk1.trace s_gtalkcap.bro

Result:

Snapshot 4.7 : Output showing Gtalk traffic

To see the captured content file less command can be used like below:

less s1.trace

Result:

```
POST /mail/channel/bind?at=xn3j34s9gsm5bi63e2y4c9fxfi99f5&VER=6&it=73003&SID=77C408F0F735F59C&RID=46858&zx=3looxtnl6xt5&t=1 H
TTP/1.1
Host: mail.google.com
User-Agent: Mozilla/5.0 (X11; U; Linux i686; en-US; rv:1.8.1.3) Gecko/20061201 Firefox/2.0.0.3 (Ubuntu-feisty)
Accept: text/xml,application/xml,application/xhtml+xml,text/html;q=0.9,text/plain;q=0.8,image/png,*/*;q=0.5
Accept-Language: en-us,en;q=0.5
Accept-Encoding: gzip,deflate
Accept-Charset: ISO-8859-1,utf-8;q=0.7,*;q=0.7
Keep-Alive: 300
Connection: keep-alive
Content-Type: application/x-www-form-urlencoded
Referer: http://mail.google.com/mail/?ui=2&view=js&name=js&ids=vsdmtnrembkr
Content-Length: 57
Cookie: gmailchat=sanmeetkbhatia@gmail.com/322221; GX=DQAAAHoAAABdCJrAeUaTiNd9ALTN7jj9oHV7MxbTC_RcJuOdmfMdvS5NAxAKyU3KD7Ae884
RLRNU25yQsB20Z4zWKcVCKKlu0I3CLjz4J__ZZ6xIAbhWpIO2-5uEjDxXZHcDi_CXugpHq06bMRu8Ji-qaEriZd_q3fSIDwGuB6xHVKe_HtXFmQ; S=gmail=rWki
2IrsH3Fy1ap2Bzkizg:gmail_yj=7n1tkyx2FZX3OIEalzA5Uw:gmproxy=U8xfWr0u8Eg:gmproxy_yj=yZ9mvPDxP2A:gmproxy_yj_sub=MYpW4pAIaNQ; GMA
IL_AT=xn3j34s9gsm5bi63e2y4c9fxfi99f5; PREF=ID=e9f145d97dfde1f2:TM=1200910601:LM=1201159574:GM=1:S=-UYHaMdNT20XGjB2; NID=7=Zq3
WXH6gzYeR0H0rZESjDfPglCYJbpWG0kitdEP9uU2QgzOOZO5rKwpqPnUskM9kKu33LgTs29vjc5FBS4Q0hkKCxj9FWICcorKzhUDhFaz6FuP_vQ-E7GJjElYq8xa5
; TZ=-330; GMAIL_RTT=286; SID=DQAAAHgAAAA14Eco7s2OydZ4WCZp6P3Xrq4o5VjIqPOzmdj43HDCWWIXqTWLjWwt9-1fIAbKSqaAtbMU4-5LwR1u8r1EOqw
3pYCTq7DFgbuCTuVf85uWKAcUPWL-tAgUAP_j6cMkLeNaMD1ZpCj8SKCVkYCG_mRTT7_mz15e9mh1bmCnhg78hw
Pragma: no-cache
Cache-Control: no-cache
```

Snapshot 4.8 Tracefile with content information of packet headers

4.3.8 Detect all packets of live GTalk traffic using a script

In the previous exercise not all information about packets but only headers information is stored. Detect all packets of live GTalk traffic using a script.

Signature File : s_gtalk.sig

```
signature gtalk-1
{
ip-proto == tcp
dst-ip == 72.14.253.125
dst-port == 5222
event "GMAIL TRAFFIC"
 }
```

Signature File : s_gtalk1.sig

```
signature gtalk-2
{
 ip-proto == tcp
 src-port == 5222
 event "GTALK Response TRAFFIC"
```

```
}
```

Script File: s_gtalklive.bro

```
redef signature_files += "s_gtalk";
redef signature_files += "s_gtalk1";
redef ignore_checksums = T;
event signature_match(state:signature_state, msg:string, data:string)
{
local id=state$id;
if(id=="gtalk-1")
{
print(msg);
}
if(id=="gtalk-2")
{
print(msg);
}
}
event connection_established(c:connection)
{
print(c$id);
set_record_packets(c$id,F);
}
```

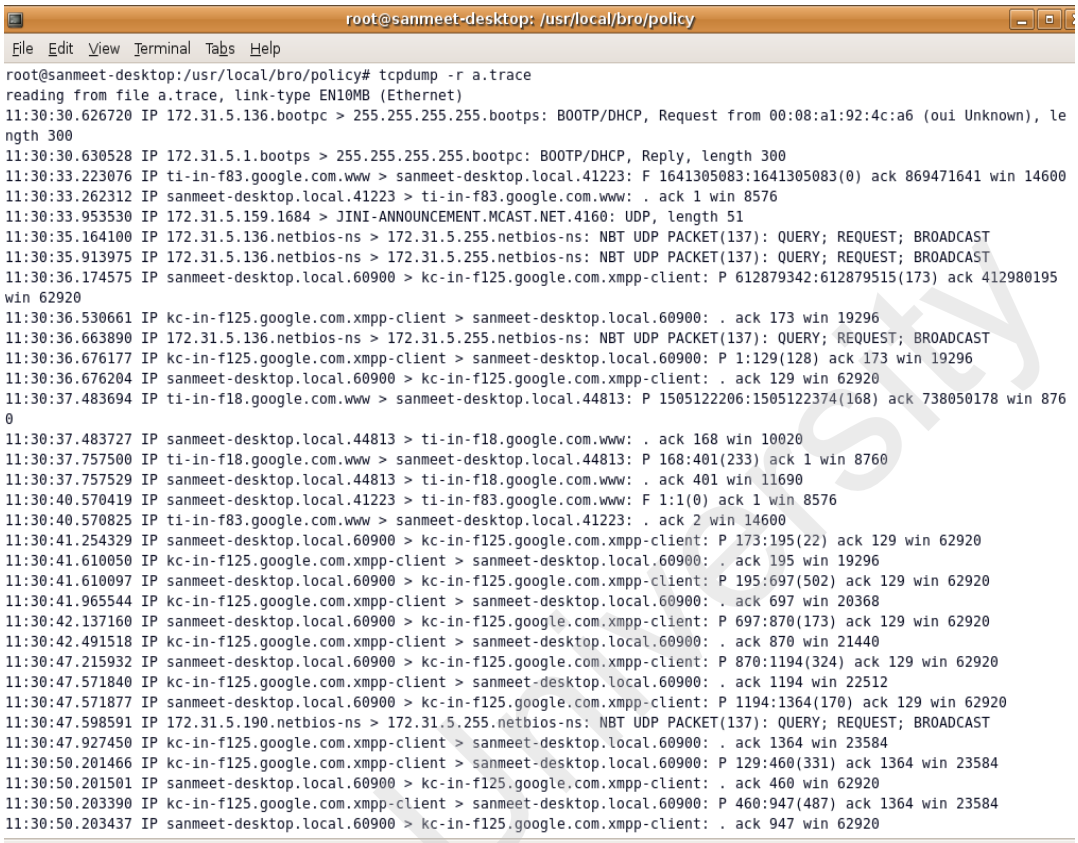
In this task no trace file is there as we have experiment it on live traffic. So *bro –i* will be used. Following is the complete command to capture the packets in a trace file:

bro -i eth0 -w a.trace s_gtalklive.bro

To see the output captured in the tracefile a.trace we can not simply use *less* command because it's a binary file. However, we can use *tcpdump* command to see the captured packets. Command to See the Captured gtalk Traffic**:**

tcpdump -r a.trace

Result:



Snapshot 4.9 Captured packets with Gtalk traffic

4.3.9 A Bro script, where the user can maintain a list of URLs, if any of the URLs are hit, log the connection to a file

Script: s_connlog.bro

```
@load weird
@load alarm
@load http
global path: string;
global urls: set[string]
= {"www.yahoo.com","mail.google.com","www.ieee.org","www.youtube.com",
"www.bro-ids.org"};
global cid: addr;
global shanz_log = open_log_file("http") &redef;
```

```
redef ignore_checksums = T;
event http_request(c: connection, method: string, original_URI: string, unescaped_URI: string, version: string)
{
    path = original_URI;
 }
event http_header(c: connection, is_orig: bool, name: string, value: string)
{
            if(name == "HOST" )
            {
              local v = edit(value," ");
              print v;
              if( v in  urls)
              {
print shanz_log,fmt(%s:%s->%s:%s,
c$id$orig_h,c$id$orig_p,c$id$resp_h,c$id$resp_p);
       }
      }
}
```

Trace File: trace.out

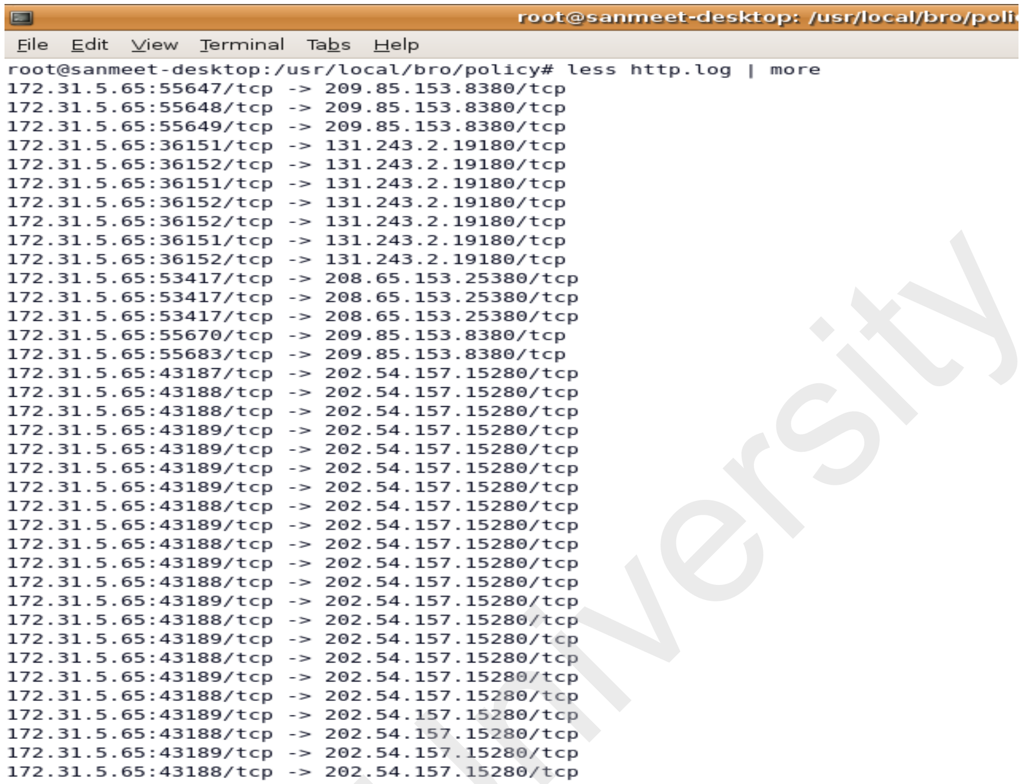
The above file is captured traffic file against which the script s_connlog.bro will be experimented. The following command will be used to run the script:

bro -r trace.out s_connlog.bro

The output is logged in a file instead of stdout. The following command is used to see that log file.

less http.log

Result:




```
root@sanmeet-desktop:/usr/local/bro/policy# less http.log | more
172.31.5.65:55647/tcp -> 209.85.153.8380/tcp
172.31.5.65:55648/tcp -> 209.85.153.8380/tcp
172.31.5.65:55649/tcp -> 209.85.153.8380/tcp
172.31.5.65:36151/tcp -> 131.243.2.19180/tcp
172.31.5.65:36152/tcp -> 131.243.2.19180/tcp
172.31.5.65:36151/tcp -> 131.243.2.19180/tcp
172.31.5.65:36152/tcp -> 131.243.2.19180/tcp
172.31.5.65:36152/tcp -> 131.243.2.19180/tcp
172.31.5.65:36151/tcp -> 131.243.2.19180/tcp
172.31.5.65:36152/tcp -> 131.243.2.19180/tcp
172.31.5.65:53417/tcp -> 208.65.153.25380/tcp
172.31.5.65:53417/tcp -> 208.65.153.25380/tcp
172.31.5.65:53417/tcp -> 208.65.153.25380/tcp
172.31.5.65:55670/tcp -> 209.85.153.8380/tcp
172.31.5.65:55683/tcp -> 209.85.153.8380/tcp
172.31.5.65:43187/tcp -> 202.54.157.15280/tcp
172.31.5.65:43188/tcp -> 202.54.157.15280/tcp
172.31.5.65:43188/tcp -> 202.54.157.15280/tcp
172.31.5.65:43189/tcp -> 202.54.157.15280/tcp
172.31.5.65:43189/tcp -> 202.54.157.15280/tcp
172.31.5.65:43189/tcp -> 202.54.157.15280/tcp
172.31.5.65:43189/tcp -> 202.54.157.15280/tcp
172.31.5.65:43188/tcp -> 202.54.157.15280/tcp
172.31.5.65:43189/tcp -> 202.54.157.15280/tcp
172.31.5.65:43188/tcp -> 202.54.157.15280/tcp
172.31.5.65:43189/tcp -> 202.54.157.15280/tcp
172.31.5.65:43188/tcp -> 202.54.157.15280/tcp
172.31.5.65:43189/tcp -> 202.54.157.15280/tcp
172.31.5.65:43188/tcp -> 202.54.157.15280/tcp
172.31.5.65:43189/tcp -> 202.54.157.15280/tcp
172.31.5.65:43188/tcp -> 202.54.157.15280/tcp
172.31.5.65:43189/tcp -> 202.54.157.15280/tcp
172.31.5.65:43188/tcp -> 202.54.157.15280/tcp
172.31.5.65:43189/tcp -> 202.54.157.15280/tcp
172.31.5.65:43188/tcp -> 202.54.157.15280/tcp
172.31.5.65:43189/tcp -> 202.54.157.15280/tcp
172.31.5.65:43188/tcp -> 202.54.157.15280/tcp
```

Snapshot 4.10: Connections attempting to access restricted URLs

Chapter 5
Conclusion and Future Work

5.1 Conclusion

Computer networks have brought the world together by bridging the information gap among people. Network technology has undergone a revolution with better and faster ways of sending information between computers. Unfortunately security systems and policies to govern these networks have not progressed as rapidly. Today's network is very complex and the whole world is focusing on ease of use and functionality. This is making us more insecure. For hackers, these 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. So there is a huge need of detecting the threats and intrusion. For this purpose number of solutions is there, IDS is one of them. Bro is the most popular and effective IDS which can be used to detect these threats.

In this thesis work we have explored and designed the policy scripts of Bro to detect various kinds of traffic like web traffic, mail traffic, web mail traffic etc. The scripts are experimented against captured traffic as well as live traffic. However no IDS can detect all the intrusions. So we need a combination of various techniques.

The Bro Scripts discussed here can be used for the purpose of Network Security. By using these scripts:

- Web traffic sent on HTTP can be analyzed.

- Email traffic using one of the well known protocols like SMTP can be analyzed.
- Webmail traffic can be analyzed.
- Instant Messenger (IM) traffic used for chat purpose supported by XMPP [20] protocol can be analyzed. However there are no inbuild signatures for this type of traffic in Bro, new signatures have been developed, by using signature matching feature of Bro, to serve the purpose.

5.2 Future Work

This thesis work can be thought of one of the modules among several other modules like Linux Packet Control, Protocol Compiler, IPS and Network Traffic Visualization, which have been taken up by my other colleagues to make an overall project for the security of networks using Deep Packet Inspection (DPI).

This module can be clubbed with the other modules in future for the overall goal of securing the network using DPI.

In this thesis work we could not experiment some kind of traffic like P2P [21], VoIP [19] and SSL [18]. P2P traffic is a relatively new class of traffic that is bane of todays networks. In future however these types of traffic can also be taken care of. Moreover other threats can also be detected and new signatures can be added.

References

1. Akhil Narayan Karkera, "Design And Implementation Of A Policy-Based Intrusion Detection System – Generic Intrusion Detection Model For A Distributed Network" , University of Florida, 2002
2. Allot Commu nications, "Digging Deeper Into Deep Packet Inspection (DPI)", White Paper, 2007
3. Bro: An Open Source Network Intrusion Detection System, Robin Sommer, Computer Science Department, TU Munchen, Germany, 2003
4. Bro Workshop 2007
   http://www.bro-ids.org/bro-workshop-2007
5. Deep Inspection, http://www.ranum.com/security/computer_security /editorials/deepinspect/index.html
6. Erricsson, "Managing Network Security", White Paper, October 2006
7. eSoft, "Modern Network Security: The Migration to Deep Packet Inspection", White Paper, 2006
8. Fengmin Gong, "Deciphering Detection Techniques: Part II Anomaly-Based Intrusion Detection", March 2003
9. Fortinet, Addressing the Limitations of Deep Packet Inspection with Complete Content Protection, White Paper, January, 2004
10. Hakan Albag, "Network & Agent Based Intrusion Detection Systems", TU Munich, Dep. of Computer Science , Istanbul Tech. Uni.
11. Hypertext Transfer Protocol-HTTP/1.1
    www.w3.org/Protocols/rfc2616/rfc2616.html
12. IntruPro IPS, "Inline Intrusion Prevention", White Paper, 2004
13. Network Protocols Handbook, Second Edition, Javvin Technologies,Inc.
14. NMAP - A Stealth Port Scanner Andrew J. Bennieston
    http://www.nmap-tutorial.com
15. Pieter de Boer & Martin Pels, "Host-based Intrusion Detection Systems", February 2005
16. Przemyslaw Kazienko, Piotr Dorosz "Intrusion Detection Systems (IDS) Classification; methods; techniques", June 2004

17. Rafeeq Ur Rehman, "Intrusion Detection Systems with Snort", Prentice Hall, New Jersey 07458
18. RFC 2246 (rfc 2246) - Secure Sockets Layer (SSL) http://www.ietf.org/rfc/rfc2246.txt
19. RFC 3265 (rfc 3265) - Voice over IP (VoIP) http://www.protocols.com/pbook/VoIPFamily.htm
20. RFC 3921 (rfc 3921) – Extensible Messaging and Presence Protocol (XMPP) http://www.xmpp.org/rfcs
21. RFC 4981 (rfc 4981) - P2P ftp://ftp.rfc-editor.org/in-notes/rfc4981.txt
22. S. McCanne, C. Leres and V. Jacobson, libpcap, available via anonymous ftp to ftp.ee.lbl.gov, 1994
23. Theuns Verwoerd , "Active Network Security", Honours Report 5 November, 1999
24. Vern Paxson, Flexible, High-Speed Intrusion Detection Using Bro, Berkley CA, USA, 2004 http://www-nrg.ee.lbl.gov/bro.html
25. Vern Paxson , Bro: A System for Detecting Network Intruders in Real-Time, Lawrence Berkeley National Laboratory
26. Wayne T Work, "Intrusion Detection Systems", Security Gauntlet Consulting,Naugatuck
27. William Stalling, Cryptography and Network Security Principles and Practices, Pearson Education, 2002
28. Wireshark, Man Pages http://www.wireshark.org/docs/man-pages/wireshark.html

List of Publications

1. Sanmeet Kaur, Dr Maninder Singh, *"Analysis of Various Intrusion Detection Techniques*", International Conference on Challenges and Developments in IT, May 2008 at Punjab College of Technical Education, Ludhiana.