Design and Development of Honeypot for Proactive Monitoring of Campus Network

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

Master of Engineering
in
Computer Science and Engineering

Submitted By
Gurpreet Singh
(Roll No. 800932008)

Under the supervision of:
Dr. Maninder Singh
Associate Professor
CSED, Thapar University

COMPUTER SCIENCE AND ENGINEERING DEPARTMENT
THAPAR UNIVERSITY
PATIALA - 147004

June 2011

Certificate

I hereby certify that the matter which is being presented in the thesis report titled, "***Design and Development of Honeypot for Proactive Monitoring of Campus Network***," in partial fulfillment of the requirements for the award of degree of Master of Engineering in *Computer Science and Engineering* submitted in Computer Science and Engineering Department of Thapar University, Patiala, is an authentic record of my own work carried out under the supervision of *Dr. Maninder Singh* and refers other researchers work which are duly listed in the reference section.

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

(Gurpreet Singh)

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

(Dr. Maninder Singh)
Associate Professor
CSED, Thapar University

Countersigned by

(Dr. Maninder Singh)
Head
Computer Science and Engineering Department
Thapar University
Patiala

11/7/21

(Dr. S. K. Mohapatra)
Dean (Academic Affairs)
Thapar University
Patiala

Acknowledgement

No volume of words is enough to express my gratitude towards my thesis supervisor **Dr. Maninder Singh**, Head of Department, Computer Science & Engineering, whose guidance, wisdom and invaluable help has aided me in the completion of thesis. He has helped me to explore numerous topics related to the thesis in an organized and methodical manner and provided me with many valuable insights into various technologies.

I would also like to thank my friends and classmates who were always there at the need of the hour and provided with all the help and facilities, which I required, for the completion of my thesis work.

Most importantly, I would like to thank my Mama, Papa, Mummy and the Almighty for showing me the way and encouraging me through the difficult times I encountered during the completion of my thesis work.

Gurpreet Singh

Abstract

Honeypots are closely monitored decoys that are employed in a network to study the trail of hackers and to alert network administrators of a possible intrusion. Honeypots are a relatively new technique for achieving network security. While other techniques for securing networks e.g. IDS, Firewall etc are made to keep the attackers out, for the first time in the history of network security there is a technique which intends to keep the attackers 'in' thus allowing the researchers to gain more insight into the workings of an attacker. A great thing about using Honeypots is that they can be easily improvised using already existing technologies. This thesis shows a similar approach where a campus honeypot is developed by joining together various separate techonologies.

In this thesis the concept and working of a virtual honeypot is discussed and a new honeypot solution ***Campus Honeypot*** is developed using various open-source techonologies and Honeyd honeypot. A virtual honeypot can emulate multiple honeypots on one physical machine, and so provide great flexibility in representing one or more networks of machines. In the process a way to overcome the OS fingerprint mismatch is shown and arpd program is patched to allow it to send unicast ARP replies and to send replies faster. In the end a sample run of the Campus Honeypot is shown and its results and output are shown.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Security

The word '*secure*' is made up of two words: *se* – meaning without or apart from and *cure* – meaning to care for, to be concerned about; thus *secure* means something without care or safe. Computer security means to the ability of a system to protect information and system resources with respect to confidentiality, integrity and availability [1].

1.2 Growth of Computer Networks

Despite its humble beginnings in the labs of DARPA the Internet today has become the backbone of today's economy. Today most the computers of the world are interconnected using networks and the Internet is a collection of such networks. The primary building blocks of the Internet are the Local Area Networks (LANs). Although the principle method of communication on Internet is TCP/IP (Transport Control Protocol/Internet Protocol) protocol suite but the Internet is fast becoming the environment with multiple protocols.

1.3 Threats to Security

Unfortunately, with the increasing use and financial importance of the Internet, computer systems and computer networks have become valuable targets for cyber criminals. Human miscreants as wells computer viruses, worms and other types of autonomously propagating malware attempt to exploit vulnerabilities in system platforms and software applications in order to gain control of as many computers as they can. Threats that arise range from innocuous TSR (Terminate and Stay Resident program) displaying an annoying message to digital identity hijacking to robbing one of its money and secret information.

To keep up with the threats raised by computer criminals computer security researchers have devised several ways to keep such miscreants away. The security implementations range from simple hosts.accepts and hosts.deny files in Unix system to large corporate solutions namely UTM (Unified Threat Management) systems. The range also consist of firewalls, IDS (Intrusion Detection System), Antivirus software, Antimalware software etc. However all the above techniques have one thing in common; they all devised to keep bad guys away. No attempt is made to study the behavior and their method of working.

However, one such set of solutions exists which focuses not in keeping bad guys away but in inciting them to attack and thus to study their working so that security professionals can know where do they need to focus. The rest of the thesis concentrates on the development of one such solution. It uses an already existing honeypot Honeyd and enhances it to make it fully automatic and easy to operate by creating a complete workflow around it and thus enable it to be used on multiple PCs at once in a corporate or academic environment.

1.4 Security Vs Functionality and Connectivity

It can be argued that a 100% secure system is impossible to attain. Security seems inversely proportional to functionality. In order to make a product more functional more amount of coding needs to be done. It is not uncommon to find several bugs per KLOC (Kilo Lines of Code) of the product. Thus more functional a product is the less secure it becomes. So the only possibility of having a security system is by putting it in a safe and then putting the safe under the ground but such a system would be of little use. Of course, one cannot afford to do this on his/her computer systems thus one must devise ways to attain as much security as possible. A similar argument can be made for Security Vs Connectivity, the more ways there are to connect to a system the more is the possibility that it will be attacked by crackers.

1.5 Security Triangle

A good security professional try to have the highest level of security in his infrastructure (ideal situation). Unfortunately, too much security controls may have a bad impact on the users, preventing them to perform their work in good conditions. Security Triangle helps to find the right balance between:

- Security
- Functionality

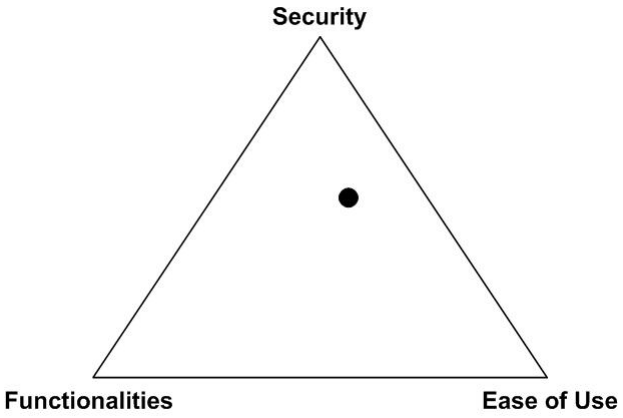

Figure 1.1: Security Triangle
[2]

- Ease of use

The Security Triangle is shown in Figure 1.1

In an ideal situation, the black point must stay at the center of the triangle, this is the best balance. If one of the components is increased, the two are affected. If security is increased (move the point nearby the top corner), the distance of black point from with functionalities and ease of use will increase. Same for the ease of use. By adding nice features to the project, security is decreased.

1.6 Elements of Security

Security is the state of well being of information and infrastructure in which the possibility of successful yet undetected theft, tempering and disruption of information and services is kept low and tolerable.

Security rests on Confidentiality, Integrity and Availability, also known as the **CIA Triad**

1.6.1 Confidentiality

Confidentiality is the concealment of information or resources. Confidentiality is the term used to prevent the disclosure of information to unauthorized individuals or systems. Confidentiality is necessary (but not sufficient) for maintaining the privacy of the people whose personal information a system holds

1.6.2 Integrity

Integrity refers to the trustworthiness of data or resources in terms of preventing improper and unauthorized changes. In information security, integrity means that data

cannot be modified undetectably. Integrity is violated when a message is actively modified in transit. Information security systems typically provide message integrity in addition to data confidentiality.

1.6.3 Availability

Availability refers to the ability to use the information or resources desired. For any information system to serve its purpose, the information must be available when it is needed. This means that the computing systems used to store and process the information, the security controls used to protect it, and the communication channels used to access it must be functioning correctly. High availability systems aim to remain available at all times, preventing service disruptions due to power outages, hardware failures, and system upgrades. Ensuring availability also involves preventing denial-of-service attacks.

1.7 Need of Security

Today security is very important because as technology is progressing the computer systems are getting more and more user friendly and hence easier to use. Today computer users range from pre-teens to senior citizens as compared to earlier when computers where only in the domain of scientists in big corporations and universities. Also in order to sustain easy of use todays computers are becoming more and more complex to administer. Thus to sum up the need of security the following arguments can be given:

1. Evolution of technology focused on easy of use.
2. Increasing complexity of computer infrastructure, administration and management.

As technology is progressing the computers are getting more and more complex and hence the ways to attack them i.e. Attack Vectors are also getting more and more complex. This can be best summed up by the graph in Figure 1.2.

The graph shows that with time as computers are getting complex the attack vectors as also getting very complex. Earlier password guessing was used to break into computers, nowadays attack vectors are shifted to more complex things like Man-In-The-Middle Distributed Denial-Of-Service (MITM DDOS) attacks.

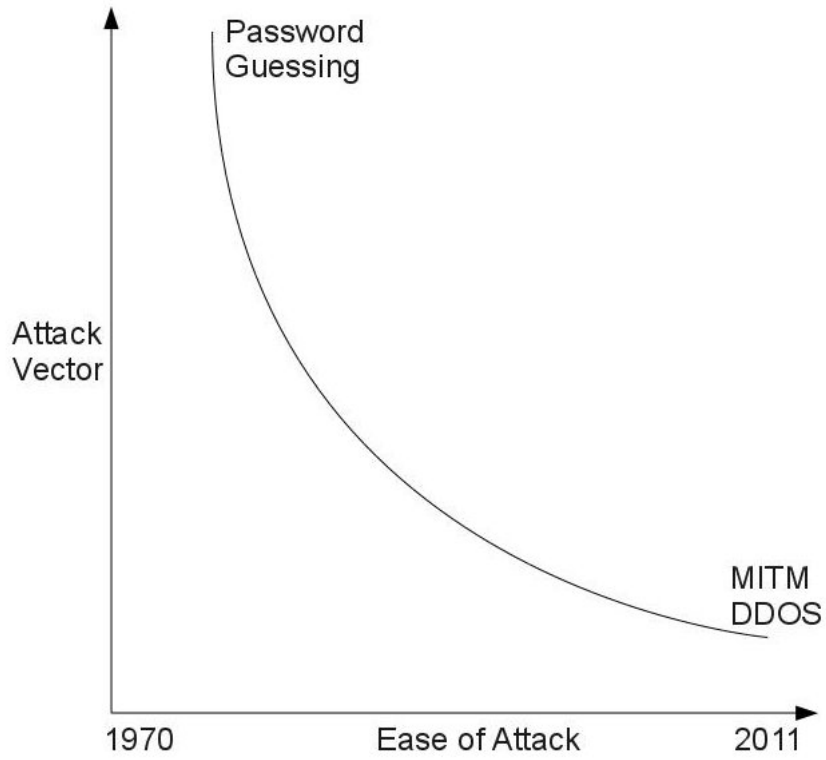

Figure 1.2: Graph depicting Ease of Attack Vs. Attack Vector with time

1.8 Hacking Terminology

Hacker

The noun ‘Hacker’ refers to a person who enjoys learning the details of computer system and stretch their capabilities.

Hacking

The verb ‘Hacking’ describes the rapid development of new programs or the reverse engineering of already existing software to make the code better and efficient.

Cracker

The term ’Cracker’ refers to a person who uses his hacking skills for offensive purposes.

Ethical Hacker

The term Ethical Hacker refers to security professionals who apply their skills for good.

Threat

Threat is an action or event that might prejudice security. A threat is a potential violation of security.

Vulnerability

Vulnerability is the existence of weakness, design or implementation error that can lead to an unexpected, undesirable event compromising the security of the system.

Target of Evaluation

Target of Evaluation is an IT system, product or component that is identified/subjected as requiring security evaluation.

Attack

Attack is an assault of system security that derives from an intelligent threat. An attack is any action that violated security.

Exploit

Exploit is a defined way to breach the security of systems through vulnerability.

1.9 Classes of Hackers

Several subgroups of the computer underground with different attitudes and aims use different terms to demarcate themselves from each other, or try to exclude some specific group with which they do not agree. The hackers can be broadly divided into following classes:

1.9.1 Black Hats

These are the individuals with extraordinary computing skills, resorting to malicious or destructive activities. They are also known as Crackers. A Black Hat Hacker is a hacker who violates computer security for little reason beyond maliciousness or for personal gain. Black Hat Hackers are the epitome of all that the public fears in a computer criminal. Black Hat Hackers break into secure networks to destroy data or make the network unusable for those who are authorized to use the network.

1.9.2 White Hats

These are individuals professing hacker skills and using them for defensive purposes. They are also known as Security Analysts. A white hat hacker breaks security for non-malicious reasons, for instance testing their own security system. This classification also includes individuals who perform penetration tests and vulnerability assessments within a contractual agreement. Often, this type of 'white hat' hacker is called an ethical hacker.

1.9.3 Grey Hats

There are individuals who work both offensively and defensively at various times. A grey hat hacker is a combination of a Black Hat and a White Hat Hacker. A Grey Hat Hacker may surf the internet and hack into a computer system for the sole purpose of notifying the administrator that their system has been hacked, for example. Then they may offer to repair their system for a small fee.

1.9.4 Elite Hacker

A social status among hackers, elite is used to describe the most skilled. Newly discovered exploits will circulate among these hackers. Elite groups such as Masters of Deception conferred a kind of credibility on their members.

1.9.5 Script Kiddie

A script kiddie is a non-expert who breaks into computer systems by using prepackaged automated tools written by others, usually with little understanding of the underlying concepthence the term script (i.e. a prearranged plan or set of activities) kiddie (i.e. kid, childan individual lacking knowledge and experience, immature)

1.9.6 Neophyte

A neophyte, "n00b", or "newbie" is someone who is new to hacking or phreaking and has almost no knowledge or experience of the workings of technology, and hacking.

1.9.7 Blue Hat

A blue hat hacker is someone outside computer security consulting firms who is used to bug test a system prior to its launch, looking for exploits so they can be closed.

1.9.8 Hacktivist

A hacktivist is a hacker who utilizes technology to announce a social, ideological, religious, or political message. In general, most hacktivism involves website defacement or denial-of-service attacks [3].

1.10 Phases of Hacking

Hackers typically approach an attack using five common phases. It is important to understand these phases of hacking attacks in order to better defend against them.

1.10.1 Reconnaissance

Before hacking online business or corporate infrastructure, hackers first perform routine and detailed reconnaissance. Hackers must gather as much information about business and networks as possible. Anything they discover about their target can be valuable during their attack phases. Strategies for hacking rely on a foundation of knowledge and understanding, arising initially from whatever the hacker can learn about their targets. Methods of reconnaissance include Dumpster Diving, Social Engineering, Google Searching, Google Hacking and work their way up to more insidious methods such as infiltrating employees environments from coffee shops to simply walking in and setting up in a cubicle and asking a lot of questions. Whatever methods are used to perform reconnaissance, hackers will usually collect a large amount of information varying from trivial to sensitive, all of which may be useful during their attacks.

1.10.2 Network and System Scanning

Probing a target's network can reveal vulnerabilities that create a hit list or triage list,for hackers to work through. Hackers may be either general hackers or specialized hackers such as phreakers, but their intent is same i.e. to access information and services that they should not gain access to. Much of the information gathered during the hacker's reconnaissance phase now come into play. In many ways,this phase of network scanning is an extension of the reconnaissance phase. Hackers want to learn more about network mapping, phone system structure,and internal informational architecture. Learning what routers, firewalls, IDS systems and other network components exist can lead hackers to beneficial hacking information by researching known vulnerabilities of known network devices. Typically, hackers perform port scans and port mapping, while attempting to discover what services and versions of services are actively available on any open or available ports.

1.10.3 Gaining Access

Open ports can lead to a hacker gaining direct access to services and possibly to internal network connections. This phase of attack is the most important and the most dangerous. Although some hack attacks do not need direct network access to damage target's business such as Denial of Services (DoS). Simple methods of attack are available to network-connected hackers including session hijacking, stack-based buffer overflow and similar security exploits. Smurf attacks try to get network users to respond and the hacker uses their real IP Addresses to flood them with problems. Whether the hacker is successful attacking an internal system has much to do with how vulnerable the specific system is, which is related to system configurations and architecture.

1.10.4 Maintaining Access

Hackers may choose to continue attacking and exploiting the target system,or to explore deeper into the target network and look for more systems and services. Not all attackers remain connected to the exploited network,but from a defensive strategy it must be expected. Hackers may deploy programs to maintain access by launching VNC clients from within target's network, providing access to external systems,opening Telnet sessions and similarly serious services like FTP and SSH,or upload rootkits and Trojans to infiltrate and exploit target's network and systems to the point where they have complete root level control. Hackers can continue to sniff network looking for more information to use against the target. Trojans can export sensitive information to hackers,such as credit card records,usernames and passwords.

1.10.5 Covering Tracks

Most hackers will attempt to cover their footprints and tracks as carefully as possible. Although not always the case, removing proof of a hacker's attacks is their best defense against legal and punitive action. It is most likely that low-end hackers and newbie hackers will get caught at a much higher rate than expert level hackers who know how to remain hidden and anonymous. Gaining root level access and administrative access is a big part of covering one's tracks as the hacker can remove log entries and do so as a privileged administrator as opposed to an unknown hacker. Placing programs inside target's network to continually send sensitive information out to anonymous drop-off points allows hackers to cover their tracks while maintaining access. Steganography allows hackers to hide information inside objects that are not obvious,such as image headers and meta tags. Tunneling allows hackers to perform their insidious work through one service that is carried over another service,to increase the difficulty of finding them.

As shown in Figure 1.3 these five phases of a hacker's attack loop back to the be-

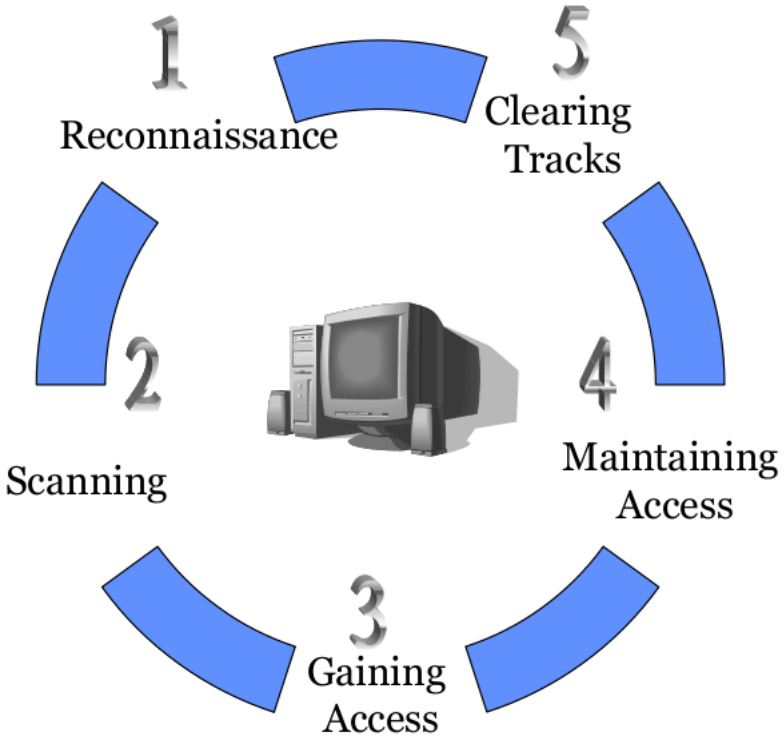

Figure 1.3: Phases of Hacking
[5]

ginning. A successful attack with maintained access often results in continuing reconnaissance. The more the hacker learns about a target's internal operations means the more likely he will be back to intrude and exploit more networks,systems,internal services,and business resources [4].

Chapter 2

Literature Survey

2.1 The Internet

The Internet has revolutionized the computer and communications world like nothing before. The invention of the telegraph, telephone, radio, and computer set the stage for this unprecedented integration of capabilities. The Internet is at once a world-wide broadcasting capability, a mechanism for information dissemination, and a medium for collaboration and interaction between individual s and their computers without regard for geographic location. The Internet represents one of the most successful examples of the benefits of sustained investment and commitment to research and development of information infrastructure. Beginning with the early research in packet switching, the government, industry and academia have been partners in evolving and deploying this exciting new technology [6].

2.2 Brief History of Internet

A network is a group of connected, communicating devices such as computers and printers. An internet is two or more networks that can communicate with each other. The most notable internet is called the Internet, a collaboration of more than hundreds of thousands interconnected networks. Private individuals as well as well as various organizations such as government agencies, schools, research facilities, corporations, and libraries in more than 100 countries use the Internet. Even though currently millions of people are using it, this amazing communication system came into being only in 1969.

2.2.1 APRANET

In the mid 1960s, mainframe computers in research organizations were stand-alone devices. Computers from different manufacturers were unable to communicate with each other. The Advanced Research Project Agency (ARPA) in the Department of defense (DOD) was interested in finding a way to connect computers together so that the researchers they funded could share their findings, thereby reducing costs and eliminating duplication of effort.

In 1967, at an Association of Computing Machinery (ACM) meeting, ARPA presented its ideas for ARPANET, a small network of connected computers. The idea was that each host computer (not necessarily from same manufacturer) would be attached to a specialized computer, called an "interface message processor" (IMP). The IMPs, in turn, would be connected to each other. Each IMP had to be able to communicate with other IMPs as well as with its own attached host.

By 1969, ARPANET was a reality. Four nodes, at the University of California at Los Angeles (UCLA), the University of California at Santa Barbara (UCSB), Stanford Research Institute (SRI), and the University of Utah were connected via the IMPs to form a network. Software called the Network Control Protocol (NCP) provided communication between the hosts.

2.2.2 Birth of the Internet

In 1972, Vint Cerf and Bob Kahn, both of whom were part of the core APRANET group, collaborated on what they called the "Internetting Project". They wanted to link different networks together so that a host on one network could communicate with a host on a second, different network. There were many problems to overcome: diverse packet sizes, diverse interfaces, and diverse transmission rates, as well as differing reliability requirements. Cerf and Kahn devised the idea of a device called a "gateway" to serve as the intermediary hardware to transfer packets from one network to another.

Figure 2.1 shows a partial map of the Internet based on the January 15, 2005 data found on opte.org. Each line is drawn between two nodes, representing two IP Addresses. The length of the lines are indicative of the delay between those two nodes. This graph represents less than 30% of the Class C networks reachable by the data collection program in early 2005 [7]. Lines are color-coded according to their corresponding RFC 1918 allocation as follows:

- Dark blue: net, ca, us
- Green: com, org
- Red: mil, gov, edu

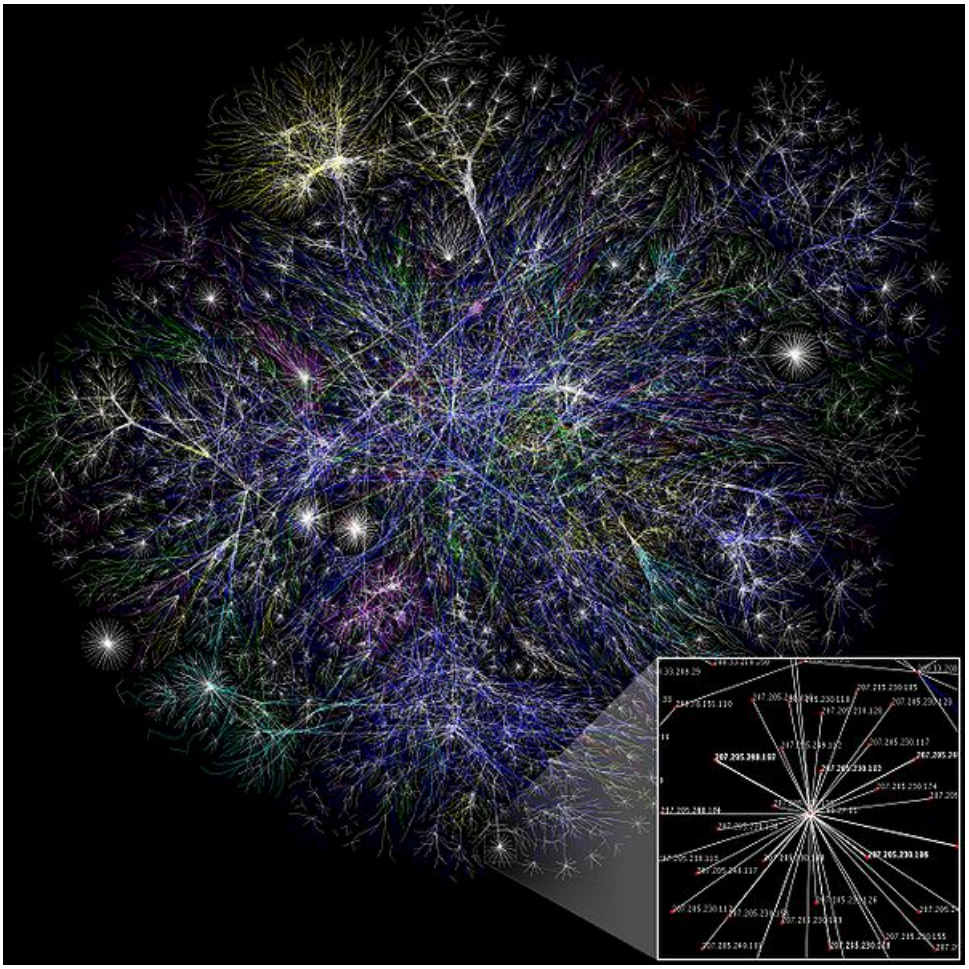

Figure 2.1: Partial Map of Internet
[7]

- Yellow: jp, cn, tw, au, de
- Magenta: uk, it, pl, fr
- Gold: br, kr, nl
- White: unknown

2.3 TCP/IP History

Cerf and Kahn's landmark 1973 paper outlined the protocols to achieve end-to-end delivery of packets. This was a new version of NCP. This paper on transmission control protocol (TCP) included concepts such as encapsulation, the datagram, and the functions of a gateway. A radical idea was the transfer of responsibility for error correction from the IMP to the host machine. This ARPA Internet now became the

focus of the communication effort. Around this time responsibility for the ARPANET was handed over to the Defense Communication Agency (DCA).

In October 1977, an internet consisting of three different networks (ARPANET, packet radio, and packet satellite) was successfully demonstrated. Communication between networks was now possible.

Shortly thereafter, authorities made a decision to split TCP into two protocols: Transmission Control Protocol (TCP) and Internetworking Protocol (IP). IP would handle datagram routing while TCP would be responsible for higher level functions such as segmentation, reassembly, and error detection. The internetworking protocol became known as TCP/IP.

In 1981, under a DARPA contract, UC Berkeley modified the UNIX operating system to include TCP/IP. This inclusion of network software along with a popular operating system did much to further the popularity of networking. The open (non-manufacture-specific) implementation on Berkeley UNIX gave every manufacturer a working code base on which they could build their products.

In 1983, authorities abolished the original ARPANET protocols and TCP/IP became the official protocol for the ARPANET. Those who wanted to use the Internet to access a computer on a different network had to be running TCP/IP [8].

2.4 OSI Model

The Open Systems Interconnection model (OSI model) was a product of the Open Systems Interconnection effort at the International Organization for Standardization. Developed in 1984 it is a way of sub-dividing a communications system into smaller parts called layers. Similar communication functions are grouped into logical layers. A layer provides services to its upper layer while receiving services from the layer below. On each layer, an instance provides service to the instances at the layer above and requests service from the layer below.

For example, a layer that provides error-free communications across a network provides the path needed by applications above it, while it calls the next lower layer to send and receive packets that make up the contents of that path. Two instances at one layer are connected by a horizontal connection on that layer.

OSI Model is now considered the primary model for interconnection of networks. OSI Model is divided into seven layers with each layer providing services to the layer above it. Each layer is self sufficient in the way that each layer can perform its task without the help of other layers. Each layer communicates with its neighboring layers using well defined access points. Each layer can be changed independently without affecting other layers. The seven layers of OSI are:

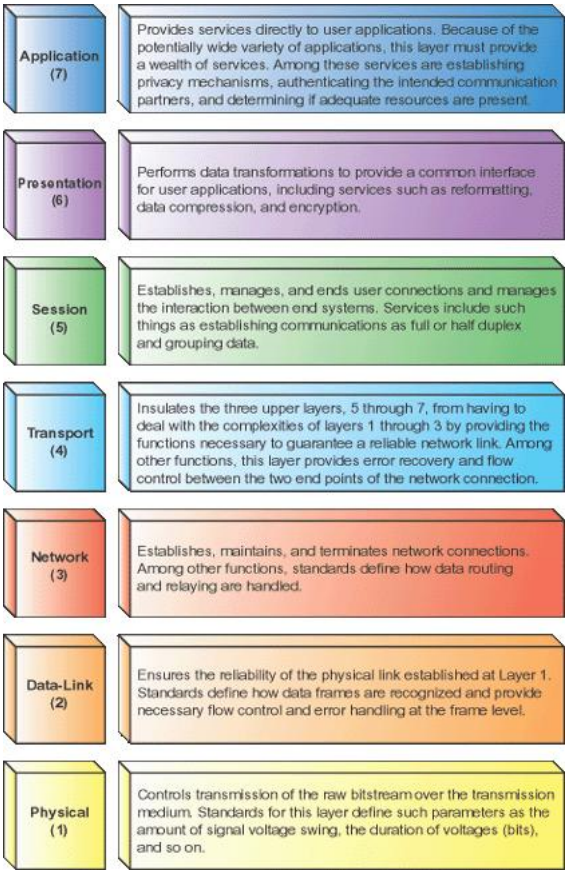

Figure 2.2: Seven Layers of OSI and their functions
[9]

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

Layer 1 and 2 are normally embedded in hardware. Layer 4, 5, 6 and 7 are normally implementated in software. Layer 3 could be either in software or embedded in hardware.

The Physical Layer defines electrical and physical specifications for devices. In particular, it defines the relationship between a device and a transmission medium, such as

a copper or optical cable. This includes the layout of pins, voltages, cable specifications, hubs, repeaters, network adapters, host bus adapters (HBA used in storage area networks) and more.

The Data Link Layer provides the functional and procedural means to transfer data between network entities and to detect and possibly correct errors that may occur in the Physical Layer.

The Network Layer provides the functional and procedural means of transferring variable length data sequences from a source host on one network to a destination host on a different network, while maintaining the quality of service requested by the Transport Layer (in contrast to the data link layer which connects hosts within the same network). The Network Layer performs network routing functions, and might also perform fragmentation and reassembly, and report delivery errors. Routers operate at this layer – sending data throughout the extended network and making the Internet possible. This is a logical addressing scheme – values are chosen by the network engineer. The addressing scheme is not hierarchical.

The Transport Layer provides transparent transfer of data between end users, providing reliable data transfer services to the upper layers. The Transport Layer controls the reliability of a given link through flow control, segmentation/segmentation, and error control. Some protocols are state- and connection-oriented. This means that the Transport Layer can keep track of the segments and retransmit those that fail. The Transport layer also provides the acknowledgment of the successful data transmission and sends the next data if no errors occurred.

The Session Layer controls the dialogues (connections) between computers. It establishes, manages and terminates the connections between the local and remote application. It provides for full-duplex, half-duplex, or simplex operation, and establishes checkpointing, adjournment, termination, and restart procedures. The OSI model made this layer responsible for graceful close of sessions, which is a property of the Transmission Control Protocol, and also for session check pointing and recovery, which is not usually used in the Internet Protocol Suite. The Session Layer is commonly implemented explicitly in application environments that use remote procedure calls.

The Presentation Layer establishes context between Application Layer entities, in which the higher-layer entities may use different syntax and semantics if the presentation service provides a mapping between them. If a mapping is available, presentation service data units are encapsulated into session protocol data units, and passed down the stack. This layer provides independence from data representation (e.g., encryption) by translating between application and network formats. The presentation layer transforms data into the form that the application accepts. This layer formats and encrypts data to be sent across a network. It is sometimes called the syntax layer.

The Application Layer is the OSI layer closest to the end user, which means that

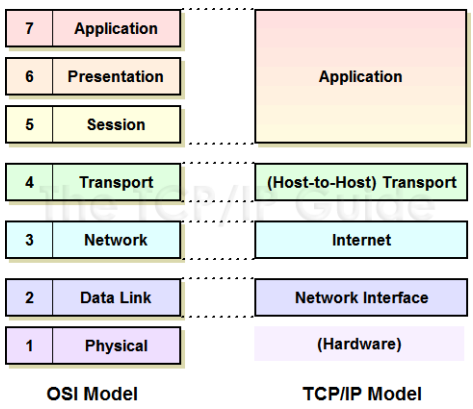

Figure 2.3: Comparison of OSI Model with TCP/IP Model
[12]

both the OSI application layer and the user interact directly with the software application. This layer interacts with software applications that implement a communicating component. Such application programs fall outside the scope of the OSI model [10].

2.5 TCP/IP Model

The TCP/IP model is a description framework for computer network protocols created in the 1970s by DARPA, an agency of the United States Department of Defense. It evolved from ARPANET, which was the world's first wide area network and a predecessor of the Internet. The TCP/IP Model is sometimes called the Internet Model or the DoD Model.

The TCP/IP model, or Internet Protocol Suite, describes a set of general design guidelines and implementations of specific networking protocols to enable computers to communicate over a network. TCP/IP provides end-to-end connectivity specifying how data should be formatted, addressed, transmitted, routed and received at the destination. Protocols exist for a variety of different types of communication services between computers [11].

TCP/IP model consist of 4 layers, with the lowest two layers of OSI Model – Physical and Data Link clubbed together to form Host-to-Network layer (Figure 2.3)

2.6 Internet Protocols

Several protocols are running on the internet (Figure 2.4), each for its specific purpose. Major protocols are described below.

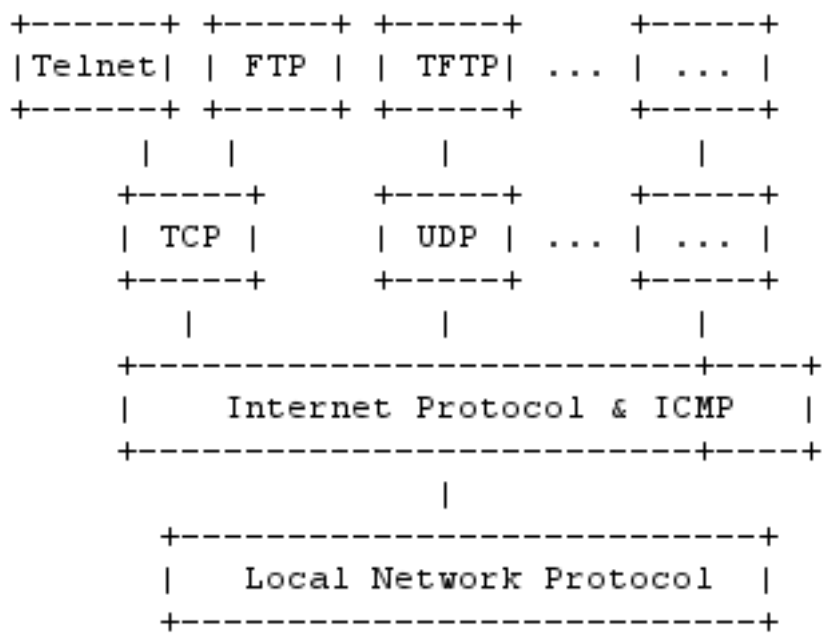

Figure 2.4: Relationship of protocols with each other
[13]

2.6.1 Internet Protocol

The Internet Protocol (IP) is designed for use in interconnected systems of packet-switched computer communication networks. Such a system has been called a "catenet". The internet protocol provides for transmitting blocks of data called datagrams from sources to destinations, where sources and destinations are hosts identified by fixed length addresses. The internet protocol also provides for fragmentation and reassembly of long datagrams, if necessary, for transmission through "small packet" networks.

The internet protocol is specifically limited in scope to provide the functions necessary to deliver a package of bits (an internet datagram) from a source to a destination over an interconnected system of networks. There are no mechanisms to augment end-to-end data reliability, flow control, sequencing, or other services commonly found in host-to-host protocols. The internet protocol can capitalize on the services of its supporting networks to provide various types and qualities of service.

The internet protocol implements two basic functions: addressing and fragmentation. The internet modules use the addresses carried in the internet header to transmit internet datagrams toward their destinations. The selection of a path for transmission is called routing. The internet modules use fields in the internet header to fragment and reassemble internet datagrams when necessary for transmission through "small packet" networks.

The first major version of IP, now referred to as Internet Protocol Version 4 (IPv4) is the dominant protocol of the Internet, although the successor, Internet Protocol Version

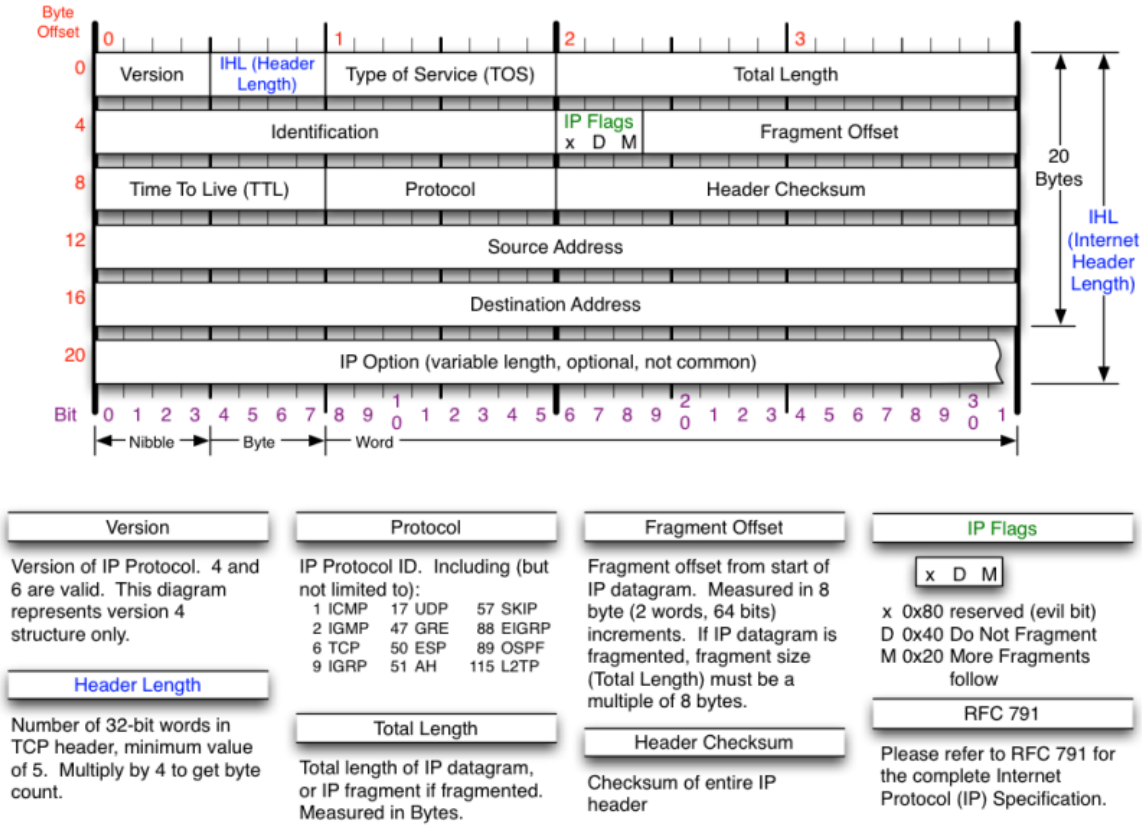

Figure 2.5: IPv4 Frame Format
[14]

6 (IPv6) is in active, growing deployment worldwide. The IPv4 header is shown in Figure 2.5 [13].

2.6.2 Transmission Control Protocol

The Transmission Control Protocol (TCP) is intended for use as a highly reliable host-to-host protocol between hosts in packet-switched computer communication networks, and in interconnected systems of such networks. TCP provides reliable, ordered delivery of a stream of bytes from a program on one computer to another program on another computer. TCP is the protocol that major Internet applications such as the World Wide Web, email, remote administration and file transfer rely on.

TCP provides a communication service at an intermediate level between an application program and the Internet Protocol (IP). That is, when an application program desires to send a large chunk of data across the Internet using IP, instead of breaking the data into IP-sized pieces and issuing a series of IP requests, the software can issue a single request to TCP and let TCP handle the IP details.

IP works by exchanging pieces of information called packets. A packet is a sequence of octets and consists of a header followed by a body. The header describes the

packet's destination and, optionally, the routers to use for forwarding until it arrives at its destination. The body contains the data IP is transmitting.

Due to network congestion, traffic load balancing or other unpredictable network behavior, IP packets can be lost, duplicated or delivered out of order. TCP detects these problems, requests retransmission of lost data, rearranges out-of-order data and even helps minimize network congestion to reduce the occurrence of the other problems. Once the TCP receiver has reassembled the sequence of octets originally transmitted, it passes them to the application program. Thus, TCP abstracts the application's communication from the underlying networking details. TCP is utilized extensively by many of the Internet's most popular applications, including the World Wide Web (WWW), E-mail, File Transfer Protocol, Secure Shell, peer-to-peer file sharing, and some streaming media applications.

TCP is optimized for accurate delivery rather than timely delivery, and therefore TCP sometimes incurs relatively long delays (in the order of seconds) while waiting for out-of-order messages or retransmissions of lost messages. It is not particularly suitable for real-time applications such as Voice over IP. For such applications, protocols like the Real-time Transport Protocol (RTP) running over the User Datagram Protocol (UDP) are usually recommended instead.

TCP is a reliable stream delivery service that guarantees delivery of a data stream sent from one host to another without duplication or losing data. Since packet transfer is not reliable, a technique known as positive acknowledgment with retransmission is used to guarantee reliability of packet transfers. This fundamental technique requires the receiver to respond with an acknowledgment message as it receives the data. The sender keeps a record of each packet it sends, and waits for acknowledgment before sending the next packet. The sender also keeps a timer from when the packet was sent, and retransmits a packet if the timer expires. The timer is needed in case a packet gets lost or corrupted.

TCP consists of a set of rules: for the protocol, that are used with the Internet Protocol, and for the IP, to send data "in a form of message units" between computers over the Internet. At the same time that IP takes care of handling the actual delivery of the data, TCP takes care of keeping track of the individual units of data transmission, called segments, that a message is divided into for efficient routing through the network. For example, when an HTML file is sent from a Web server, the TCP software layer of that server divides the sequence of octets of the file into segments and forwards them individually to the IP software layer (Internet Layer). The Internet Layer encapsulates each TCP segment into an IP packet by adding a header that includes (among other data) the destination IP address. Even though every packet has the same destination address, they can be routed on different paths through the network. When the client program on the destination computer receives them, the TCP layer (Transport Layer)

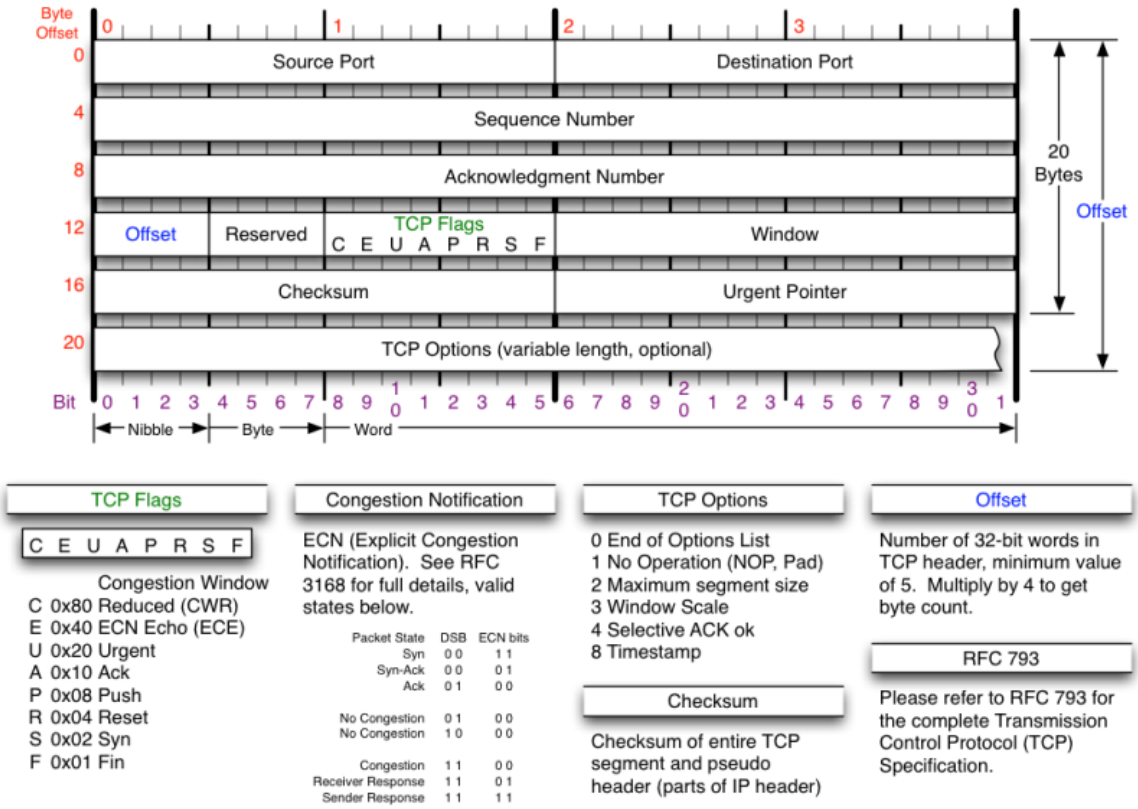

Figure 2.6: TCP Frame Format
[14]

reassembles the individual segments and ensures they are correctly ordered and error free as it streams them to an application. TCP frame format is shown in Figure 2.6 [15].

2.6.3 User Datagram Protocol

This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) is used as the underlying protocol.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [16].

UDP is one of the core members of the Internet Protocol Suite, the set of network protocols used for the Internet. With UDP, computer applications can send messages, in this case referred to as datagrams, to other hosts on an Internet Protocol (IP) network

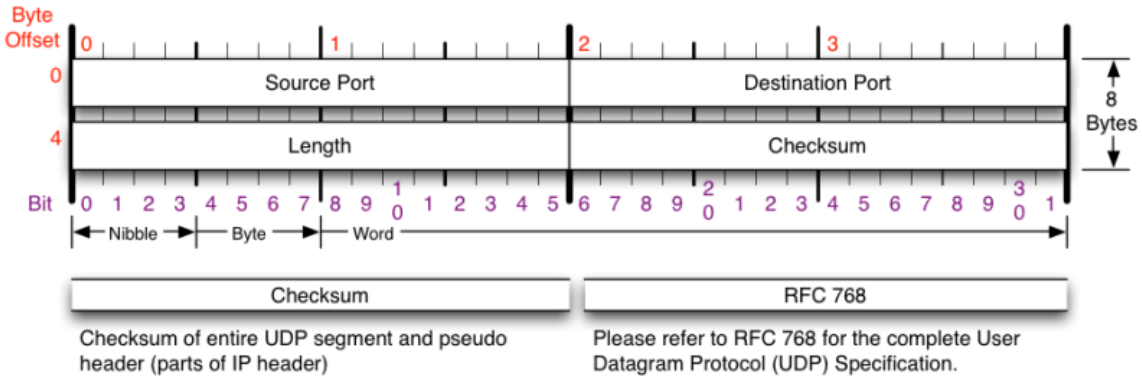

Figure 2.7: UDP Frame Format
[14]

without requiring prior communications to set up special transmission channels or data paths. The protocol was designed by David P. Reed in 1980 and formally defined in RFC 768.

UDP uses a simple transmission model without implicit handshaking dialogues for providing reliability, ordering, or data integrity. Thus, UDP provides an unreliable service and datagrams may arrive out of order, appear duplicated, or go missing without notice. UDP assumes that error checking and correction is either not necessary or performed in the application, avoiding the overhead of such processing at the network interface level. Time-sensitive applications often use UDP because dropping packets is preferable to waiting for delayed packets, which may not be an option in a real-time system. If error correction facilities are needed at the network interface level, an application may use the Transmission Control Protocol (TCP) or Stream Control Transmission Protocol (SCTP) which are designed for this purpose.

UDP's stateless nature is also useful for servers answering small queries from huge numbers of clients. Unlike TCP, UDP is compatible with packet broadcast (sending to all on local network) and multicasting (send to all subscribers).

Common network applications that use UDP include: the Domain Name System (DNS), streaming media applications such as IPTV, Voice over IP (VoIP), Trivial File Transfer Protocol (TFTP) and many online games [17]. The UDP frame format is shown in Figure 2.7.

2.6.4 Internet Control Message Protocol

The Internet Protocol (IP) is used for host-to-host datagram service in a system of interconnected networks called the Catenet. The network connecting devices are called Gateways. These gateways communicate between themselves for control purposes via a Gateway to Gateway Protocol (GGP). Occasionally a gateway or destination host will

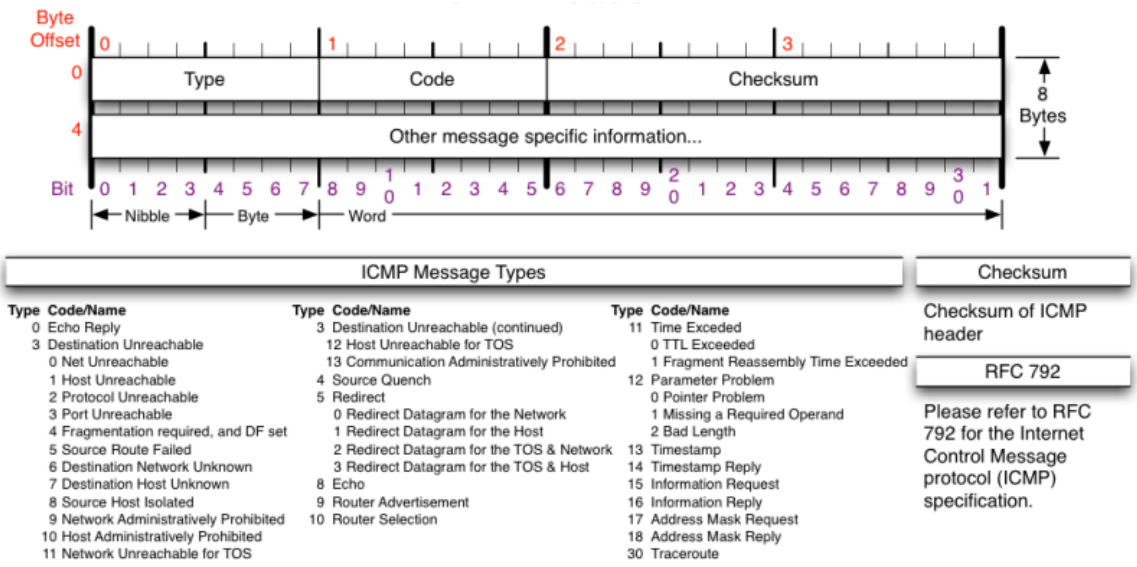

Figure 2.8: ICMP Frame Format
[14]

communicate with a source host, for example, to report an error in datagram processing. For such purposes this protocol, the Internet Control Message Protocol (ICMP), is used. ICMP, uses the basic support of IP as if it were a higher level protocol, however, ICMP is actually an integral part of IP, and must be implemented by every IP module [18].

ICMP differs from transport protocols such as TCP and UDP in that it is not typically used to exchange data between systems, nor is it regularly employed by end-user network applications (with the exception of some diagnostic tools like ping and traceroute). ICMP messages are typically generated in response to errors in IP datagrams (as specified in RFC 1122) or for diagnostic or routing purposes. ICMP errors are always reported to the original source IP address of the originating datagram [19]. The ICMP frame format is shown in Figure 2.8.

2.7 Network Intrusion Detection Systems

The goal of an Intrusion Detection System (IDS) is to identify, preferably in real time, unauthorized use, misuse, and abuse of computer systems by both system insiders and external penetrators . An IDS is used as an alternative (or a complement) to building a shield around the network. The shielding approach is deficient in several ways, including failure to prevent attacks from insiders. While IDS technology is progressing, methods to circumvent IDSs are becoming more prevalent. In light of these attacks as well as the growing prevalence of encrypted communication, alternatives such as honeypots have become more popular.

2.8 Honeypot: an Offensive Security Measure

Almost all security technologies being used today are made to keep the bad guys out. For example firewalls are made to keep systems secure by keeping bad guys from outside out of the network and keeping internal bad guys reaching out of the network, similarly IDS are designed to keep harmful attacks out of network/system. However, honeypot is a technology which is designed specifically to make the bad guys come inside the network so that their behavior can be studied and which will be used to further strengthen systems. Thus one can say that firewalls and IDS used defensive security whereas honeypots work as offensive security measure. They entice the bad guys to attack the system in order to study them.

The exact definition of a honeypot is contentious, however most definitions are some form of the following:

> *A honeypot is an information system resource whose value lies in unauthorized or illicit use of that resource.*

What this definition means is honeypots derive their value from threats using them. If the enemy does not interact or use the honeypot, then it has little value. This is very different from most security mechanisms. For example, the last thing one want an attacker to do is interact with the firewall, IDS sensor, or PKI certificate authority. Honeypots are very different, and it is this difference that makes them such a powerful tool in ones arsenal [20].

A more practical, but more limiting, definition is given by pcmag.com:

> *A server that is configured to detect an intruder by mirroring a real production system. It appears as an ordinary server doing work, but all the data and transactions are phony. Located either in or outside the firewall, the honeypot is used to learn about an intruder's techniques as well as determine vulnerabilities in the real system*

In practice, honeypots are computers which masquerade as unprotected. The honeypot records all actions and interactions with users. Since honeypots don't provide any legitimate services, all activity is unauthorized (and possibly malicious). Honeypots are analogous to the use of wet cement for detecting human intruders [21].

First, honeypots do not solve a specific problem. Instead, they are a highly flexible tool that has many applications to security. They can be used everything from slowing down or stopping automated attacks, capturing new exploits to gathering intelligence on emerging threats or early warning and prediction. Second, honeypots come in many different shapes and sizes. They can be everything from a Windows program that emulates common services, such as the Windows honeypot KFSensor3, to entire networks

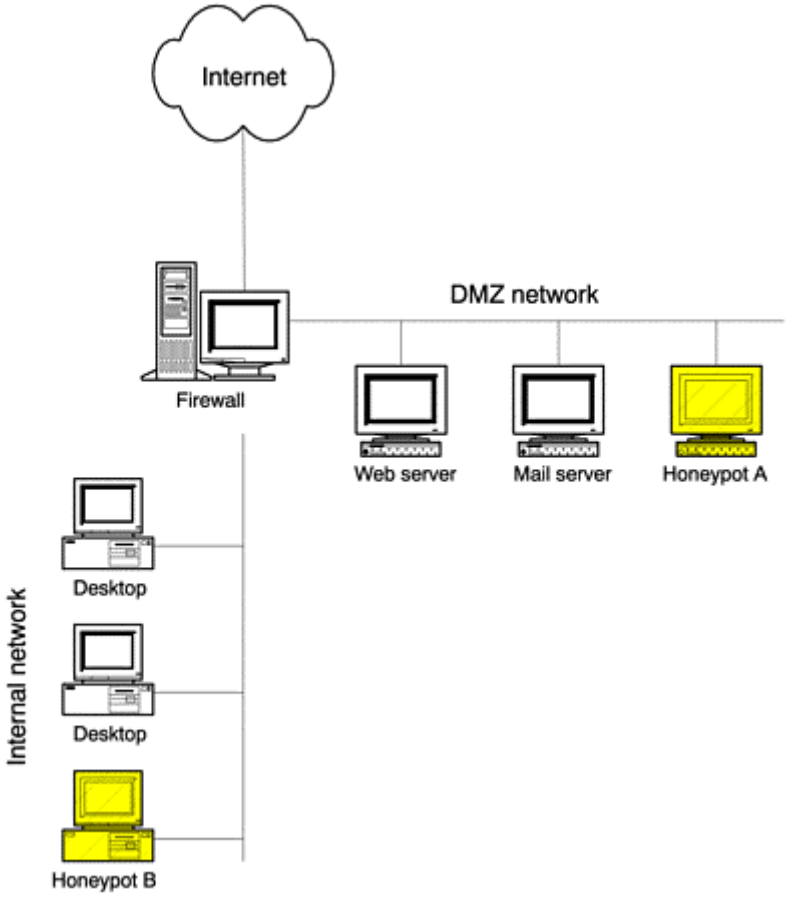

Figure 2.9: Two Honeypots

of real computers to be attacked, such as Honeynets. In fact, as will be discussed later, honeypots don't even have to be a computer, instead they can be a credit card number, Excel spread sheet, or login and password (commonly called honeytokens).

All honeypots share the concept; they are a resource or entity with no production value. By definition, a honeypot should not see any activity. Anything or anyone interacting with the honeypot is an anomaly, it should not be happening. Most likely, it implies an unauthorized or malicious activity. For example, a honeypot could be nothing more then a webserver deployed in a DMZ network. The webserver is not used for production purposes, its does not even have an entry in DNS, its merely physically located with other webservers. Any interaction with the honeypot is assumed unauthorized and most likely malicious. If the webserver honeypot is probed by external systems from the Internet, it means a probe or attack, most likely the same one other production webservers are facing. If the honeypot is probed by one of the production webservers on the DMZ, that can imply that the production webserver has been compromised by an attacker, and is now being used as a launching pad to compromise other systems on the DMZ [20].

The Figure 2.9 shows two honeypots (Honeypot A and B) deployed by an orga-

nization. Honeypot A is running on a DMZ network along with other servers of the organization. Honeypot B is running on the internal network along with other systems of the users. The intent of Honeypot A is to determine if there is any unauthorized activity happening within the DMZ. Since this system is not functioning as a normal server, no one should be connecting to it. Now, if anyone from the Internet connects to the honeypot, he is most likely probing or potentially attacking the system, perhaps looking for vulnerable Exchange servers. If any of the production servers on the DMZ, such as the new mail server or the Web server, make a connection to the honeypot, these servers may have been compromised, and now the attacker is probing for other vulnerable systems, including probing the honeypot. The system A is a honeypot by definition.

Lets suppose La Brea tarpit is installed on Honeypot B. La Brea Tarpit is a honeypot solution that detects connection attempts to a system that does not exist and then forges replies to the attacker on behalf of the system that does exist. Since the honeypot is monitoring IP address space that has no live systems, it knows that any packets sent to these non existing systems is most likely a probe or an attack. The forged replies sent by the honeypot are crafted to maintain an open connection with the attacker, slowing down or even stopping the automated attack. Honeypot B responds on behalf of systems that do not exist, so it knows anything sent to these systems is most likely an attack. Honeypot B's primary goal is not to detect attacks but slow them down, potentially even stopping them. This can give an organization critical time in responding to worms once they have been infected.

2.9 Honeypot Vs Firewall

A honeypot is quite different from a firewall. A firewall is designed to keep the bad guys out of the network whereas honeypots are designed to entice the hackers to attack the system so that a security researcher can know how hackers operate once they have access of the system or a security analyst in a corporation can know which systems and ports the hackers are most interested in.

Also firewalls log activities to all the systems of a corporation, so in case of an event it becomes very difficult to shift through all the logs in order to find a particular event log as logs also contains events related to production systems. However in case of honeypot, the logs are only due to non-productive systems, these are the systems that noone should be interacting with. So if a firewall log contain 1000 entries of all the systems of the network the honeypot's log only contain only 5-10 entries each of which is important for a researcher.

2.10 Honeypot Vs IDS

The amount of useful information provided by NIDS is decreasing in the face of ever more sophisticated evasion techniques and an increasing number of protocols that employ encryption to protect network traffic from eavesdroppers. NIDS also suffer from high false positive rates that decrease their usefulness even further. Honeypots can help with some of these problems.

A honeypot is a closely monitored computing resource that one intend to be probed, attacked, or compromised. The value of a honeypot is determined by the information that can be obtained from it. Monitoring the data that enters and leaves a honeypot lets us gather information that is not available to NIDS. For example, one can log the key strokes of an interactive session even if encryption is used to protect the network traffic. To detect malicious behavior, NIDS require signatures of known attacks and often fail to detect compromises that were unknown at the time it was deployed. On the other hand, honeypots can detect vulnerabilities that are not yet understood. For example, one can detect compromise by observing network traffic leaving the honeypot even if the means of the exploit has never been seen before.

Because a honeypot has no production value, any attempt to contact it is suspicious. Consequently, forensic analysis of data collected from honeypots is less likely to lead to false positives than data collected by NIDS [22].

IDS and firewalls are the traditional approaches to network security. The goal of an Intrusion Detection System (IDS) is to "identify, preferably in real time, unauthorized use, misuse, and abuse of computer systems by both system insiders and external penetrators." An IDS is used as an alternative (or a complement) to building a shield around the network. The shielding approach is deficient in several ways, including failure to prevent attacks from insiders. IDS often depend upon signature matching or statistical models to identify attacks. This means that unknown or novel threats may not be detected. In contrast, honeypots are designed to capture all known and unknown attacks directed against them. Because any network activity related to the honeypot represents an anomaly, even the stealthiest activity will register on a honeypot's radar. Because honeypots operate at the host level, encrypted or non-IPv4 communications that can often blind traditional network based sensors can still be captured.

2.11 Classification of Honeypots

Honeypots can be classified into various categories depending on their purpose i.e. either they are being setup for research or production (as a security measure in a corporation). Depending on the level of interaction they provided to the attacker – can the attack get full control of the honeypot or can it only connect to it and depending on

how they are implemented i.e. are they implemented on a real environment or they are running on a virtual environment as a fake system.

2.11.1 Classification according to Purpose

Honeypots can be used for production or research. Production honeypots protect an organization, while research honeypots are used to learn.

2.11.1.1 Production Honeypots

Production honeypots are what most people typically think of as a honeypot. They add value to the security of a specific organization and help mitigate risk. These honeypots are implemented within the organization because they help secure its environment, such as detecting attacks. Production honeypots are the law enforcement of honeypot technologies. Their job is to deal with the bad guys. Commercial organizations often use production honeypots to mitigate the risk of attackers. Production honeypots usually are easier to build and deploy than research honeypots because they require less functionality. Because of their relative simplicity, they generally have less risk. It's much more difficult to use a production honeypot to attack and harm other systems. However, they also generally give us less information about the attacks or the attackers than research honeypots do. One may learn which systems attackers are coming from or what exploits they launch, but he will most likely not learn how they communicate among each other or how they develop their tools. e.g. Spectre. Figure 2.10 shows a production honeypot deployed on a DMZ.

2.11.1.2 Research honeypots

Research honeypots are designed to gain information about the blackhat community. These honeypots do not add direct value to a specific organization. Their primary mission is to research the threats organizations may face, such as who the attackers are, how they are organized, what kind of tools they use to attack other systems, and where they obtained those tools. If production honeypots are similar to law enforcement, then one can think of research honeypots as counterintelligence, used to gain information on the bad guys. This information helps in understanding who the threats are and how they operate.

Research honeypots are far more involved than production honeypots. To learn about attackers, one need to give them real operating systems and applications with which to interact. This gives the researchers far more information than simply what applications are being probed. One can potentially learn who the attackers are, how they communicate, or how they develop or acquire their tools. However, this increased

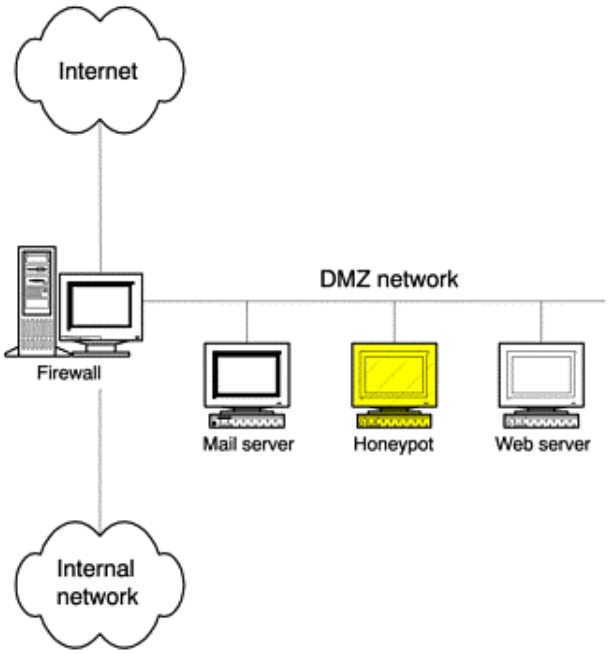

Figure 2.10: Network Diagram of a production honeypot deployed on a DMZ to detect attacks

functionality has its disadvantages. Research honeypots are more complex, have greater risk, and require more time and effort to administer. In fact, research honeypots could potentially reduce the security of an organization, since they require extensive resources and maintenance. e.g. Honeyd [23].

2.11.2 Classification according to Interaction

Honeypots can also be classified according to level of interaction they provide to an adversary.

2.11.2.1 High-interaction honeypots

High-interaction honeypots offer the adversary a full system to interact with. This means that the honeypot does not emulate any services, functionality, or base operating systems. Instead, it provides real systems and services, the same used in organizations today. Thus, the attacker can completely compromise the machine and take control of it. This allows the researcher to learn more about the tools, tactics, and motives of the attacker and get a better understanding of the attacker community. Although these types of honeypots can give deep insights into the routine procedures of an attacker, High-interaction honeypots can be time-consuming to implement.

High-interaction honeypots have some risk. The attacker can abuse a honeypot he has compromised and start to attack other systems on the Internet. This could cause both legal or ethical problems. Therefore, one need to safeguard the whole setup to

mitigate risk e.g. a Windows system running on VMWare.

High interaction honeypots (HIH) provide entire operating systems running real applications for attackers to interact with. Services are no longer emulated. Instead, real computers are deployed for attackers to remotely compromise. HIH provide great advantages, in that attackers can not only probe the honeypot but can actually break in and run command shells and applications or down-load new exploits and toolkits. Because all services are real, HIH can detect novel activities. They can gather extensive detail about an attacker's actions, including keystrokes and interactive sessions, which increase the opportunity to analyze potential threats.

The downside to HIH is that higher levels of interaction increase the level of risk. Attackers have more potential to use the honeypot for malicious purposes. Constraining attackers and controlling outbound data are the greatest challenges for honeynet operators.

2.11.2.2 Low-interaction honeypots

Low-interaction honeypots are easy to set up. Even without much experience, a network of hundreds of low-interaction virtual honeypots can be set up in a short time. A low-interaction honeypot provides only limited access to the operating system. By design, it is not meant to represent a fully featured operating system and usually cannot be completely exploited. As a result, a low-interaction honeypot is not well suited for capturing zero-day exploits. Instead, it can be used to detect known exploits and measure how often a network gets attacked. The term low-interaction implies that an adversary interacts with a simulated environment that tries to deceive him to some degree but does not constitute a fully fledged system. A low-interaction honeypot often simulates a limited number of network services and implements just enough of the Internet protocols, usually TCP and IP, to allow interaction with the adversary and make him believe he is connecting to a real system. e.g. Honeyd, LaBrea [24].

Low interaction honeypots (LIH) generally only emulate network services and host systems. LIH systems are generally limited to known threats, because emulated services don't usually respond correctly to previously unknown attacks. In any case, a determined assailant will often be able to quickly identify an LIH system. Both configuration and logging are relatively simple, and an attacker's options are normally limited to reduce

2.11.2.3 Medium-interaction honeypots

A medium-interaction honeypot might more fully implement the HTTP protocol to emulate a well-known vendor's implementation, such as Apache. However, the definition of low-interaction honeypots captures the functionality of medium-interaction

honeypots in that they only provide partial implementation of services and do not allow typical, full interaction with the system as high-interaction honeypots. Medium-interaction honeypots offer attackers more ability to interact than do low-interaction honeypots but less functionality than high-interaction solutions. They can expect certain activity and are designed to give certain responses beyond what a low-interaction honeypot would give. For example, perhaps there is a worm scanning for specific IIS vulnerabilities. A honeypot could be built to imitate a Microsoft IIS Web server, including the additional functionality that normally accompanies the application. The emulated IIS Web server could then be customized to present whatever functionality or behavior the specific worm was looking for. Whenever an HTTP connection was made to the honeypot, it would respond as an IIS Web server, giving the attacker the opportunity to interact with actual IIS functionality. This level of interaction is greater than the low-level honeypot, which would have most likely simply presented an HTTP banner. In the case of the worm, the intent is for it to attack the honeypot so one can capture the worm payload for future analysis. However, the worm has not been given a full operating system with which to interact, limiting risk. There is only an emulated application. As such this would not be a high level of interaction [23].

2.11.3 Classification according to Implementation

Apart from purpose and level of interaction, Honeypots can be classified according to their implementation i.e. if they are physical or virtual.

A physical honeypot is a real machine on the network with its own IP address. If say 10 physical honeypots are required then 10 computers in the network are required. Example of Physical honeypots running on a network is a Honeynet. Physical honeypots are often high-interaction, so allowing the system to be compromised completely, they are expensive to install and maintain. For large address spaces, it is impractical or impossible to deploy a physical honeypot for each IP address. In that case, one need to deploy virtual honeypot .

A virtual honeypot is simulated by another machine that responds to network traffic sent to the virtual honeypot. A single virtual honeypot can simulate thousands of honeypots on a network. Example of virtual honeypot is Honeyd by Neils Provos

2.12 Advantages of Honeypots

Honeypots have several advantages over firewalls and IDSes. They are enumerated as under:

1. **Small Data Sets**

   Honeypots only collect data when someone or something is interacting with them.

Organizations that may log thousands of alerts a day with traditional technologies will only log a hundred alerts with honeypots. This makes the data honeypots collect much higher value, easier to manage and simpler to analyze.

2. **Reduced False Positives**
   One of the greatest challenges with most detection technologies is the generation of false positives or false alerts. It's similar to the story of the 'boy who cried wolf'. The larger the probability that a security technology produces a false positive the less likely the technology will be deployed. Honeypots dramatically reduce false positives. Any activity with honeypots is by definition unauthorized, making it extremely efficient at detecting attacks.

3. **Catching False Negatives**
   Another challenge of traditional technologies is failing to detect unknown attacks. This is a critical difference between honeypots and traditional computer security technologies which rely upon known signatures or upon statistical detection. Signature-based security technologies by definition imply that "someone is going to get hurt" before the new attack is discovered and a signature is distributed. Statistical detection also suffers from probabilistic failures – there is some non-zero probability that a new kind of attack is going to go undetected. Honeypots on the other hand can easily identify and capture new attacks against them. Any activity with the honeypot is an anomaly, making new or unseen attacks easily stand out.

4. **Encryption**
   It does not matter if an attack or malicious activity is encrypted, the honeypot will capture the activity. As more and more organizations adopt encryption within their environments (such as SSH, IPsec, and SSL) this becomes a major issue. Honeypots can do this because the encrypted probes and attacks interact with the honeypot as an end point, where the activity is decrypted by the honeypot.

5. **IPv6 Support**
   Honeypots work in any IP environment, regardless of the IP protocol, including IPv6. IPv6 is the new IP standard that many organizations, such as the Department of Defense, and many countries, such as Japan, are actively adopting. Many current technologies, such as firewalls or IDS sensors, cannot handle Ipv6.

6. **Highly Flexible**
   Honeypots are extremely adaptable, with the ability to be used in a variety of environments, everything from a Social Security Number embedded into a database, to an entire network of computers designed to be broken into.

7. **Minimal Resources Required**

   Honeypots require minimal resources, even on the largest of networks. A simple, aging Pentium computer can monitor literally millions of IP addresses. Virtual honeypots means deploying multiple honeypots using just one machine. This can be done by using any virtualization software or using scripts to simulate a network of computers. A wonderful example of this is honeyd by Neils Provos where one can simulate multiple computers with a myriad number of operating systems and services.

8. **Simple Deployment and Low Cost**

   As a security resource, Honeypots are one of the most simple and inexpensive tool that one can use to augment his network security. Since a Honeypot is not used a production system, has little activity and does not take in exhaustive amounts of information like IDS firewalls, there is no need to use high production systems to implement and deploy. In fact, depending on deployment architecture, it could just be any old PC that isnt being used anymore.

9. **Return on Investment**

   When firewalls successfully keep attackers out, they become victims of their own success. Management may begin to question the return on their investment, as they perceive there is no longer a threat: "We invested in and deployed a firewall three years ago, and we were never attacked. Why do we need a firewall if we have never been hacked?" The reason they were never hacked is the firewall helped reduce the risk. Investments in other security technologies, such as strong authentication, encryption, and host-based armoring, face the same problem. These are expensive investments, costing organizations time, money, and resources, but they can become victims of their own success. In contrast, honeypots quickly and repeatedly demonstrate their value. Whenever they are attacked, people know the bad guys are out there. By capturing unauthorized activity, honeypots can be used to justify not only their own value but investments in other security resources as well. When management perceives there are no threats, honeypots can effectively prove that a great deal of risk does exist.

10. **Data Value**

    ne of the challenges the security community faces is gaining value from data. Organizations collect vast amounts of data every day, including firewall logs, system logs, and Intrusion Detection alerts. The sheer amount of information can be overwhelming, making it extremely difficult to derive any value from the data. Honeypots, on the other hand, collect very little data, but what they do collect is normally of high value. The honeypot concept of no expected production activity

dramatically reduces the noise level. Instead of logging gigabytes of data every day, most honeypots collect several megabytes of data per day, if even that much. Any data that is logged is most likely a scan, probe, or attack – information of high value. Honeypots can give the precise information in a quick and easy-to-understand format. This makes analysis much easier and reaction time much quicker. This is similar to a microscope effect. Whatever data it capture is placed under a microscope for detailed scrutiny [20, 25].

2.13 Disadvantages of Honeypots

Apart from all the advantages, honeypots also have some disadvantages. Disadvantages of honeypots are listed below:

1. **Risk**
   Honeypots are a security resource the bad guys to interact with, there is a risk that an attacker could use a honeypot to attack or harm other non-honeypot systems. This risk varies with the type of honeypot used. For example, simple honeypots such as KFSensor have very little risk. Honeynets, a more complex solution, have a great deal of risk. The risk levels are variable for different kinds of honeypot deployments. The usual rule is that the more complicated the deception, the greater the risk. Honeypots that are high-interaction such as Gen I Honeynets are inherently more risky because there is an actual computer involved.

2. **Limited Field of View**
   Honeypots only see or capture that which interacts with them. They are not a passive device that captures activity to all other systems. Instead, they only have value when directly interacted with. In many ways honeypots are like a microscope. They have a limited field of view, but a field of view that gives them great detail of information.

3. **Discovery and Fingerprinting**
   Though risk of discovery of a honeypot is small for script kiddies and worms, there is always a chance that advanced blackhats would be able to discover the honeypot. A simple mistake in the deception is all a savvy attacker needs to "fingerprint" the honeypot. This could be a misspelled word in one service emulation or even a suspicious looking content in the honeypot. The hacker would be able to flag the honeypot as "dangerous" and in his next attacks, he would most certainly bypass the honeypot. In fact, armed with the knowledge, an advanced blackhat could even spoof attacks to the honeypot thus redirecting attention while he attacks other vulnerable systems in the network [20, 26].

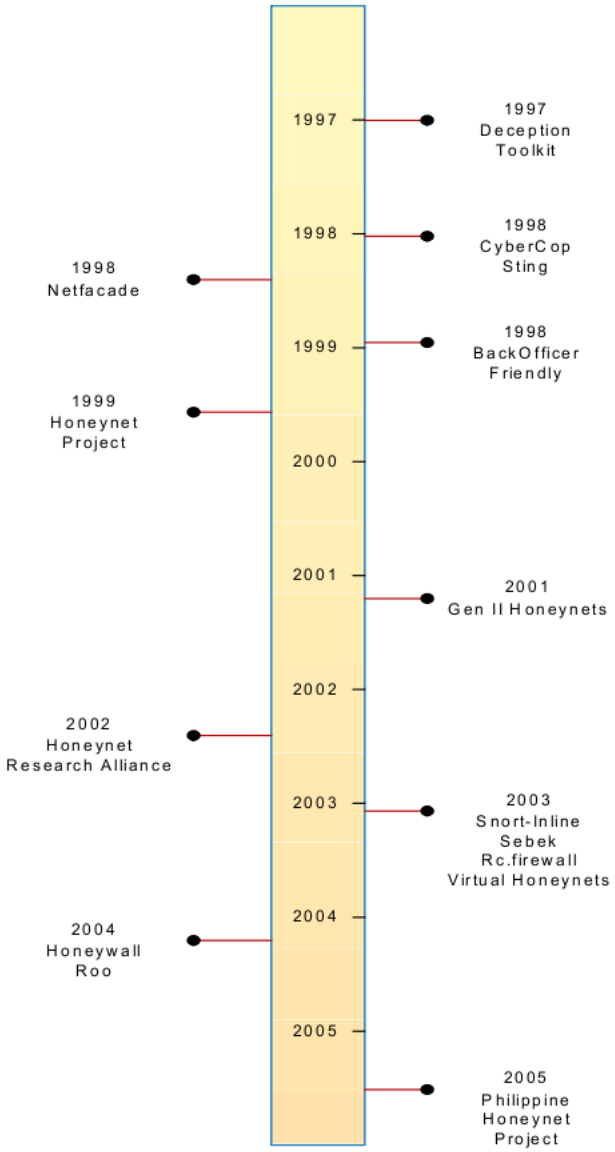

Figure 2.11: Timeline of Honeypot's Origin
[27]

2.14 Honeypot Origins

Honeypots have a unique history. Even though the concepts have been around for more than a decade, only recently have commercial products been developed or papers published on the concept (Figure 2.11).

The first resource was a book written by Clifford Stoll titled The Cuckoo's Egg [28]. The second is the whitepaper "An Evening with Berferd in Which a Cracker Is Lured, Endured, and Studied" [29], by the security icon Bill Cheswick.This does not mean that honeypots were not invented until 1990; they were undoubtedly developed and used by a variety of organizations well before then. 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.

In The Cuckoo's Egg, Clifford Stoll discusses a series of true events that occurred over a ten-month period in 1986 and 1987. Stoll was an astronomer at Lawrence Berkeley Lab who worked with and helped administer a variety of computer systems used by the astronomer community. A 75-cent accounting error led him to discover that an attacker, code named "Hunter," had infiltrated one of his systems. Instead of disabling the attacker's accounts and locking him out of the system, Stoll decided to allow the attacker to stay on his system. His motives were to learn more about the attacker and hunt him down. Over the following months he attempted to discover the attacker's identity while at the same time protecting the various government and military computers the attacker was targeting. Stoll's computers were not honeypots; they were production systems used by the academic and research communities. However, he used the compromised systems to track the attacker in a manner very similar to the concept of honeypots and honeypot technologies. Stoll's book is not technical; it reads more like a Tom Clancy spy novel. What makes the book unique and important for the history of honeypots are the concepts Stoll discusses in it.

The most fascinating thing in the book is Stoll's approach to gaining information without the attacker realizing it. For example, he creates a bogus directory on the compromised system called SDINET, for Strategic Defense Initiative Network. He wanted to create material that would attract the attention of the attacker. He then filled the directory with a variety of interesting-sounding files. The goal was to waste the attacker's time by compelling him to look through a lot of files. The more time he spent on the system, the more time authorities had to track down the attacker. Stoll also included documents with different values. By observing which particular documents the attacker copied, he could identify the attacker's motives. For example, 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.

Bill Cheswick's paper "An Evening with Berferd in Which a Cracker Is Lured, Endured, and Studied" was released in 1990. This paper is more technical than The Cuckoo's Egg. It was written by security professionals for the security community. Like The Cuckoo's Egg, everything in Cheswick's paper is nonfiction. However, unlike the book, Cheswick builds a system that he wants to be compromisedfirst documented case of a true honeypot. In the paper, he discusses not only how the honeypot was built and used but how a Dutch hacker was studied as he attacked and compromised a variety of systems. Cheswick initially built a system with several vulnerabilities (including Sendmail) to determine what threats existed and how they operated. His goal was not to capture someone specific but rather to learn what threatening activity was happening

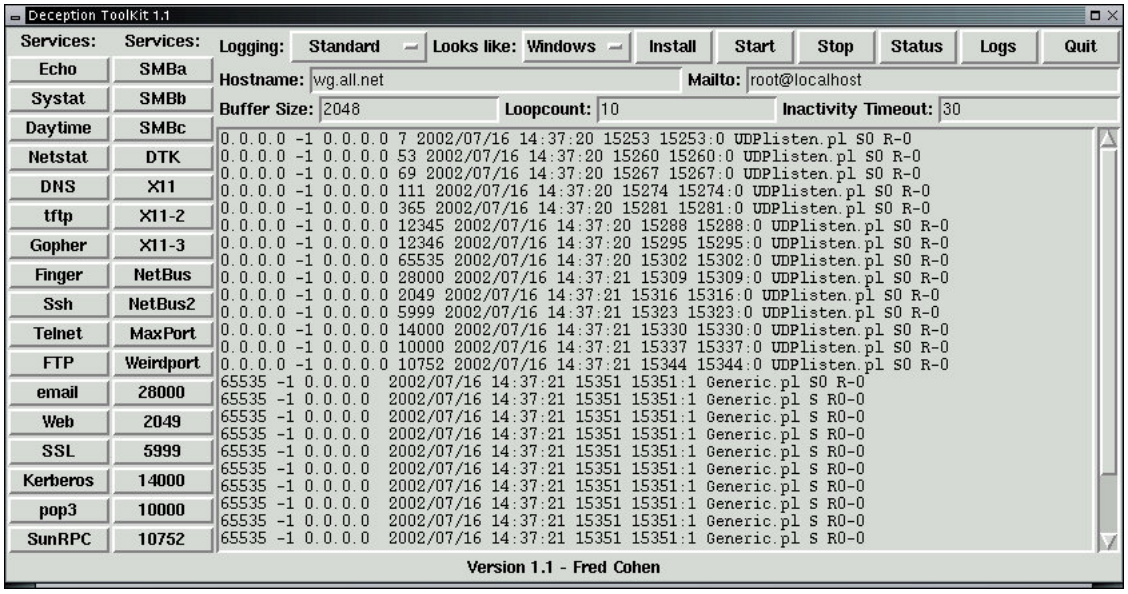

Figure 2.12: Deception Toolkit
[30]

on his networks and systems.

Cheswick's paper explains not only the different methodologies he used in building his system (he never calls it a honeypot) but also how these methodologies were used. In addition to a variety of services that appeared vulnerable, he created a controlled environment called a "jail," which contained the activities of the attacker. He writes step by step on how an intruder (called Berferd) attempts to infiltrate the system and what Cheswick was able to learn from the attacker.

The first public honeypot solution, called Deception Toolkit (DTK) (Figure 2.12), was developed by Fred Cohen. Version 0.1 was released in November 1997, seven years after The Cuckoo's Egg and "An Evening with Berferd." DTK is one of the first free honeypot solutions one could download, install, and try. It is a collection of PERL scripts and C code that is compiled and installed on a Unix system. DTK is similar to Bill Cheswick's Berferd system in that it emulates a variety of known Unix vulnerabilities. When attacked, these emulated vulnerabilities log the attacker's behavior and actions and reveal information about the attacker. The goal of DTK is not only to gain information but to deceive the attacker and psychologically confuse her. DTK introduced honeypot solutions to the security community.

Following DTK, in 1998, development began on the first commercial honeypot product, CyberCop Sting. Originally developed by Alfred Huger at Secure Networks Inc., it was purchased by NAI in 1998. This honeypot had several features different from DTK. First, it ran on Windows NT systems and not Unix. Second, it could emulate different systems at the same time, specifically a Cisco router, a Solaris server, and

an NT system.

Thus, CyberCop Sting could emulate an entire network, with each system having its own unique services devoted to the operating system it was emulating. It would be possible for an attacker to scan a network and find a variety of Cisco, Solaris, and NT systems. The attacker could then Telnet to the Cisco router and get a banner saying the system was Cisco, FTP to the Solaris server and get a banner saying the system was Solaris, or make a HTTP connection to the NT server. Even the emulated IP stacks were modified to replicate the proper OS. This way if active fingerprinting measures were used, such as Nmap, the detected OS would reflect the services for that IP address. The multiple honeypot images created by a single CyberCop Sting installation greatly increased the chance of the honeypots being found and attacked. This improved detection of and alerting to the attacker's activity.

As a commercial product it never really took off and has now been discontinued. Since its demise, several excellent commercial honeypot products have been released, including NetSec's Specter and Recourse's Mantrap .

In 1998, Marty Roesch, while working at GTE Internetworking, began work on a honeypot solution for a large government client. Roesch and his colleagues developed a honeypot system that would simulate an entire class C network, up to 254 systems, using a single host to create the entire network. Up to seven different types of operating systems could be emulated with a variety of services. Although the resulting commercial product, NetFacade, has seen little public exposure, an important side benefit of this honeypot solution is that Roesch also developed a network-based debugging tool, which eventually led to his OpenSource IDS, Snort .

The year 1998 also saw the release of BackOfficer Friendly, a Windows and Unix-based honeypot developed by Marcus Ranum and released by Network Flight Recorder. What made BOF unique is that it was free, extremely easy to use, and could run on any Windows based desktop system. All one had to do was download the tool, install it on his system, and he instantly had his own personal honeypot. Though limited in its capabilities, BOF was many people's first introduction to the concepts of honeypot technologies.

In 1999 the Honeynet Project was formed. A nonprofit research group of 30 security professionals, this group is dedicated to researching the blackhat community and sharing what they learned. Their primary tool for learning is the Honeynet, an advanced type of honeypot. Over several years the Honeynet Project demonstrated the capabilities and value of honeypots, specifically Honeynets, for detecting and learning about attacks and the attackers themselves [23].

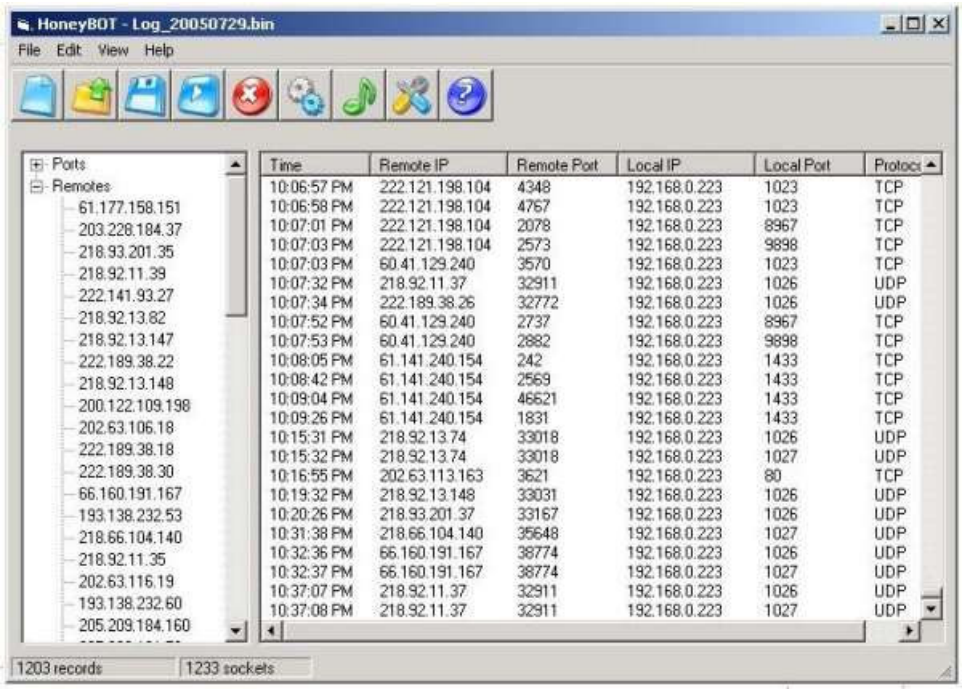

Figure 2.13: HoneyBOT User Interface
[31]

2.15 Survey of Various Honeypots

2.15.1 HoneyBOT

HoneyBOT is a Windows medium-interaction honeypot by Atomic Software Solutions . It originally began as an attempt to detect by the Code Red and Nimda worms in 2001 and has been released for free public use since 2005. HoneyBOT allows attackers to upload files to a quarantined area in order to detect trojans and rootkits. HoneyBOT's user interface is shown in figure 2.13.

HoneyBOT works by opening over 1000 UDP and TCP listening sockets on the host computer and these sockets are designed to mimic vulnerable services. When an attacker connects to these services they are fooled into thinking they are attacking a real server. The honeypot safely captures all communications with the attacker and logs these results for future analysis. Should an attacker attempt an exploit or upload a rootkit or trojan to the server the honeypot environment will safely store these files on the computer for analysis and submission to antivirus vendors

2.15.2 BackOfficer Friendly

BackOfficer Friendly, often called BOF (Figure 2.14) is one of the simplest honeypots to use. Its functionality is remarkably easy to understand and configure. BOF is a low-interaction honeypot. It runs on either Windows or Unix, although it is pri-

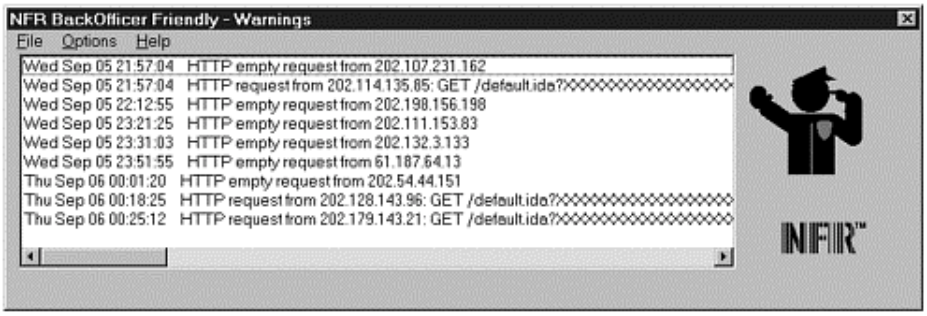

Figure 2.14: BOF honeypot detecting, logging, and alerting to a series of attacks captured in the wild

marily deployed on Windows-based systems. BOF can run on almost any Windows based-platform, including Windows 95 and Windows 98, making it an excellent solution for desktop systems, including home users. BOF can monitor up to seven emulated services. There is almost no customization options for BOF, so while it is simple to use and deploy, it is extremely restricted in its functionality. One of the best features of BOF is its price: It's free. Unlike most of the solutions are and will be discussed, BOF was not originally intended to be a honeypot. Instead, it was a tool designed as a response to a specific threat. It was first developed in 1998 by Marcus Ranum and the folks at Network Flight Recorder in response to the Cult of the Dead Cow's, known as cDc, release of Back Orifice.

BOF is a low-interaction, production honeypot. It adds value to an organization primarily by detecting and alerting to attacks. There are seven preconfigured services on which BOF can detect attacks. When a connection is made to any one of these seven services, the attempt is logged, and an alert is generated. BOF has some emulation capability, but it is extremely limited. None of the services emulate a specific application or version, only the functionality of the service. For example, the Web server does emulate a Web server; it captures attempts to get a Web page.

However, it does not emulate a specific Web server, such as Apache or IIS. This functionality limits BOF to primarily a detection technology. Also, since additional services cannot be added or modified, BOF is limited to detecting attacks only on the seven ports that BOF monitors. If an attack is made against any other port, BOF is blissfully unaware of any malicious activity. This is a limitation common to most low-interaction honeypots [23].

2.15.3 Specter

Specter's authors describes Specter as a "honeypot-based intrusion detection system" (Figure 2.15) . However, the product is primarily a honeypot designed to lure attackers away from production systems and collect evidence against the attackers.

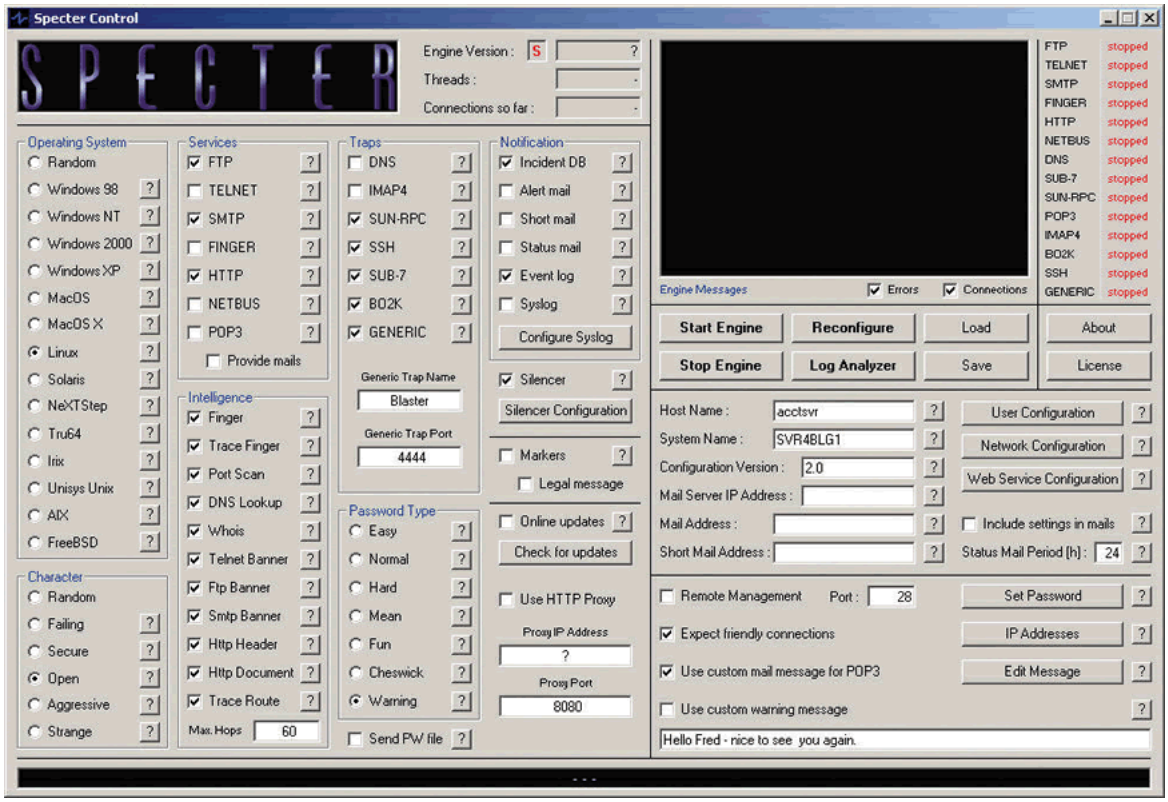

Figure 2.15: Specter's control center

Specter has a few interesting features not found in other solutions:

- Specter makes decoy data available for attackers to access and download. These data files leave marks on the attacker's computer as evidence
- Specter can emulate machines in different states: a badly configured system, a secured system, a failing system (with hardware or software failures), or an unpredictable system.
- Specter actively attempts to collect information about each attacker.

Like all honeypots, Specter operates on the principle that it serves no practical applications; therefore any Internet-based activity taking place on it is unauthorized. Anytime there is any interaction with the honeypot, this is by definition most likely malicious activity. This makes it extremely effective for detection, especially on internal networks. It will quickly tell if a bad guy has penetrated the perimeter defenses, or if one have an employee or vendor looking where they shouldn't. Specter works by listening on specific TCP services. When an attacker interacts with one of these services, Specter captures all of their activity, logs the behavior, and generates an alert. Specter currently does not have the capability to detect ICMP, UDP, or any non-standard IP

Traps	Services
DNS	FTP
IMAP4	TELNET
SUN-RPC	SMTP
SSH	FINGER
SUB-7	HTTP
BOK2	NETBUS
Generic	POP3

Table 2.1: Specter's Traps and Services

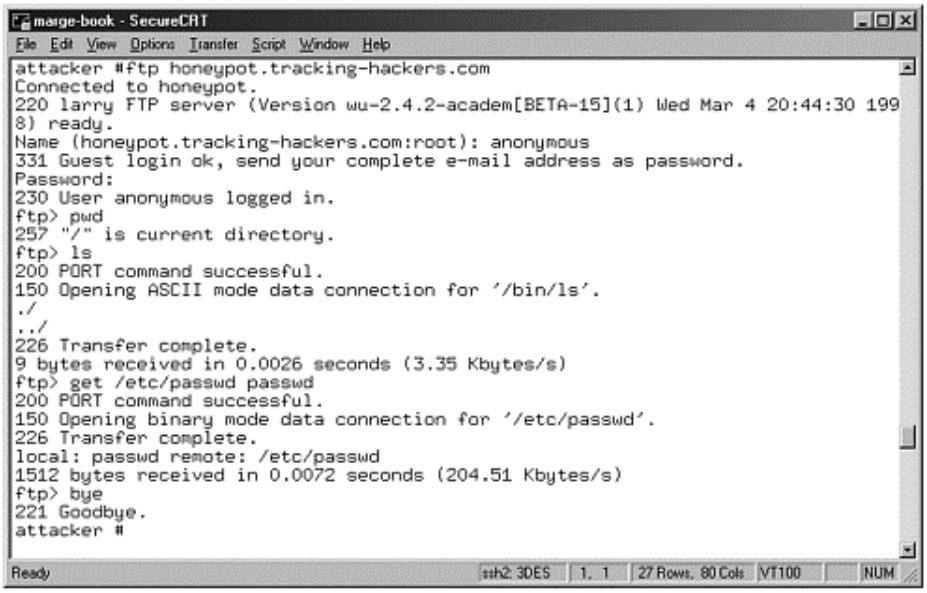

Figure 2.16: Specter emulating a vulnerable FTP server

traffic. Specter monitors the IP assigned to the computer. Unlike some other honeypots, Specter cannot monitor unused IP space. This limits the number of IPs that can be monitored with it.

Currently Specter can monitor a total of 14 TCP ports. Of these 14 monitored ports, 7 are what Specter calls traps, the other 7 are what Specter calls services. Traps are nothing more then port listeners: when a connection is made, the attempt is terminated, and then logged. Services are more advanced; they actually interact with the attacker, emulating the application. The level of emulation depends on each service. For example, the HTTP service emulates a simple Web server with default static Web pages. The FTP server allows an attacker to log in and execute some basic FTP commands [32]. The 7 traps and 7 services are listed in the Table 2.1. Figure 2.16 shows Specter emulating a vulnerable FTP server.

2.15.4 Honeyd

Honeyd is developed and maintained by Niels Provos of the University of Michigan. It is designed as a low-interaction solution; there is no operating system intended for an attacker to gain access to, only emulated services. Honeyd is designed primarily as a production honeypot, used to detect attacks or unauthorized activity. As an OpenSource solution, one can highly customize the honeypot, such as designing new emulated services. This means this honeypot can listen on any port, with as little or as much emulation as required.

Honeyd introduces several exciting new concepts to honeypots. First, it does not detect attacks against its own IP address, as both BOF and Specter do. Instead, Honeyd assumes the identity of any IP address that does not have a valid system. When an attacker attempts to connect to a system that does not exist, Honeyd receives the connection attempt, assumes the identity of the nonexistent system, and then replies to the attacker. From this point on Honeyd communicates to the attacker as the victim system. Honeyd can monitor millions of nonexistent IP addresses for connections. It can also simultaneously assume the IP addresses of thousands of victims and actively interact with attackers. Honeyd can emulate different operating systems at the same time. While Specter can emulate 13 different operating systems, it can only do one system at one time. Honeyd can emulate many different systems at the same time.

Another new concept is that Honeyd emulates an operating system not only at the application level but at the IP stack level as well. For example, if one select his Honeyd system to emulate a Windows NT 4.0 Server SP5-SP6 server, it not only emulates the services, such as an IIS Web server, but also the IP stack. In this case, if one use Nmap to fingerprint the IP stack, the response will be that the IP stack is Windows NT 4.0 Server SP5-SP6.

As a low-interaction honeypot, Honeyd is primarily a production honeypot, used to detect attackers. Its model for detection is the same as most low-interaction honeypots. When a connection is made to a port it is listening on, that connection is logged, the attacker's activity is captured, and an alert is generated. Because the services listening on the ports have some level of emulation, one can capture the attacker's interaction with the service, similar to Specter. However, Honeyd has two advantages that increase its value. The first is that it can detect connections on any TCP port. The emulated services are not required for detection; they exist only for interaction with attackers and to gain more information. This makes deployment of Honeyd simpler, and it increases its detection capabilities.

The second advantage to Honeyd is that one can modify the services emulated and the level of interaction. As an OpenSource solution, people throughout the security community can contribute code to Honeyd, including these emulated services .

Honeyd is explained in a great detail in a forth coming section as the honeypot solution the author developed uses Honeyd as its honeypot [23].

2.15.5 Deception Toolkit

The Deception ToolKit (DTK) was the first commercial honeypot (Figure 2.17). Developed by Fred Cohen, it is a toolkit designed to give defenders a couple of orders of magnitude advantage over attackers.

In the case of DTK, the deception is intended to make it appear to attackers as if the system running DTK has a large number of widely known vulnerabilities. DTK's deception is programmable, but it is typically limited to producing output in response to attacker input in such a way as to simulate the behavior of a system which is vulnerable to the attackers method. This has a few interesting side effects:

- It increases the attacker's workload because they can't easily tell which of their attack attempts works and which fail. For example, if an attack produces what appears to be a Unix password file, the attacker would normally run "Crack" to try to break into the system. But if the password file is a fake, it consumes the attackers time and effort to no result.

- It allows researchers to track attacker attempts at entry and respond before they come across a vulnerability they are susceptible to. For example, when the attacker tries to use a known Sendmail attack against a site, DTK record all of their entries to track their techniques. With this deception in place, there is no problem picking up port scans, password guessing, and all manner of other attack attempts as they happen [33].

2.15.6 Cybercop Sting

CyberCop Sting simulates a network of routers and workstations and monitors all activity sent to its virtual hosts, acting as a decoy to lure potential attackers while tracking the origin of suspicious interest in the network. While watching all traffic destined to hosts in its virtual network, CyberCop Sting performs IP fragmentation reassembly and TCP stream reassembly on the packets destined to these hosts, convincing snoopers of the legitimacy of the secret network theyve discovered.

CyberCop Sting creates a decoy, virtual TCP/IP network on a single Windows NT server or workstation. This server is attached to a larger network environment. Even if it is not advertised, a potential attacker discovers the decoy subnetwork only by running profiling or scanning tools from within the larger network; alternatively, one can

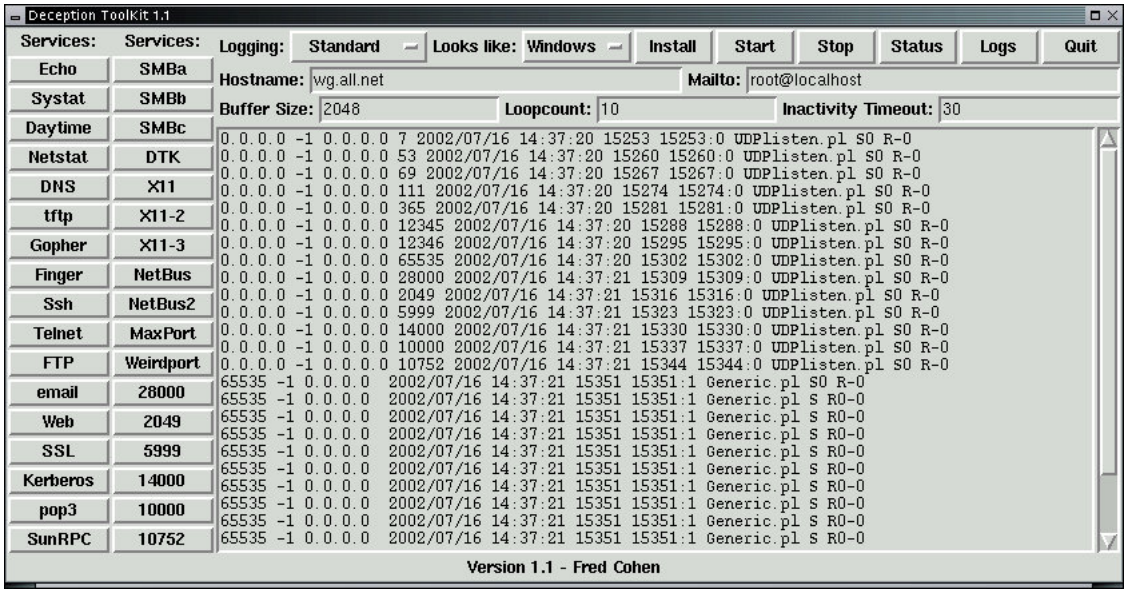

Figure 2.17: Deception Toolkit
[30]

publicize it heavily, even including it in the list of DNS entries for the domain, and lure potential attackers who might otherwise go unnoticed.

CyberCop Sting can simulate a network containing:

- Windows NT 4.0 servers
- Solaris 2.6 servers
- Cisco IOS 11.2 routers

Each virtual server or router has a real IP address and can receive and send genuine-looking packets from and to the larger network environment. Each virtual network node can run simple simulated daemons, such as finger and FTP, to further emulate the activity of a genuine system.

If a machine in the larger network environment connects to any of the fake machines inside its virtual domain, CyberCop Sting returns realistic IP stack information as it logs the IP address and port number of the machine that made the request. The attacker is lured into a “honey pot,” thinking theyve discovered unprotected information on actual machines with unique IP addresses, while CyberCop Sting records all intrusive activity and writes it to a log file. By creating a virtual network of decoy routers and servers, CyberCop Sting alerts administrators to potential attackers without putting genuine systems and data at risk [34].

2.15.7 Netfacade

NetFacade creates a Honeynet (simulated network) of hosts, with simulated IP addresses, running seemingly vulnerable services, so all traffic to the Honeynet is considered suspicious. A scan of the simulated IP addresses will return information on the simulated services as if they were real services running on actual hosts. The activity identified by NetFacade provides a clear indication an intruder attempted to compromise real network or systems components, so all traffic on the Honeynet is is logged and brought to the attention of the Security Administrator(s). This approach minimizes the amount of data reported and reduces the time required for analysis as well as reducing the frequency of false positive reports.

NetFacade provides the ability to configure security alerts at various levels and define specific actions to be taken when an alert is received. There is no limit on the number and types of attacks NetFacade is able to identify, including new and unknown attacks to enable administrators to program repeatable signatures for other intrusion detection systems and firewalls. Configuration and monitoring is through a Web-based interface, accessible only by specified workstations and users, controlled through Secure Socket Layer authentication methods. Alerts are delivered via email to a pre-defined list of recipients [35].

2.15.8 ManTrap

ManTrap is a high-interaction commercial honeypot created, maintained, and sold by Recourse Technologies. ManTrap is unique in that it is designed to be not only attacked but also compromised. ManTrap creates a highly controlled operating environment that an attacker can interact with. creates a fully functional operating system containing virtual cages rather than a limited operating system (Figure 2.18). The cages are logically controlled environments from which the attacker is unable to exit and attack the host system. However, instead of creating an empty cage and filling it with certain functionality ManTrap creates cages that are mirror copies of the master operating system. Each cage is a fully functional operating system that has the same capabilities as a production installation.

This approach creates a very powerful and flexible solution. Each cage is its own virtual world with few limitations. An administrator can customize each cage as he would a physically separate system. He can create users, install applications, run processes, even compile his own binaries. When an intruder attacks and gains access to a cage, to the attacker it looks as if the cage is a truly separate physical system. He is not aware that he is in a caged environment where every action and keystroke is recorded. Unlike jailed environments, the attacker has a complete set of binaries, devices, and system libraries with which he can interact, just as he would find on a normal system.

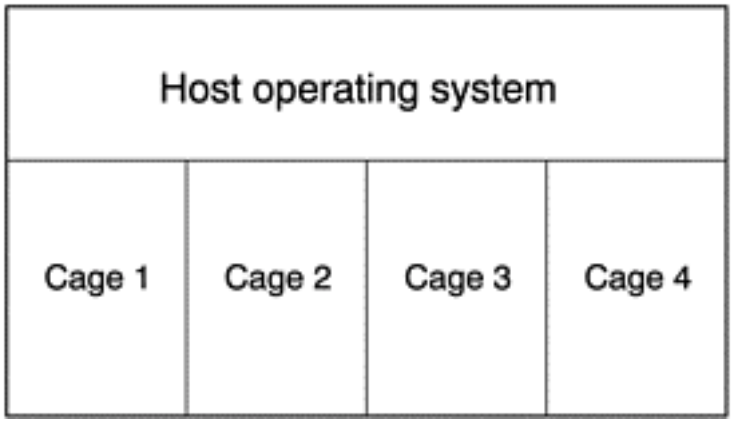

Figure 2.18: A ManTrap host with four logical cages

However, like a jailed environment, the master operating system can monitor and control the attacker.

ManTrap runs real operating systems with legitimate applications. There may be vulnerabilities within the honeypot, depending on what applications are running and which version. Attackers may be deceived into attacking and potentially compromising these systems. Since ManTrap is a high-interaction honeypot, attackers can potentially waste a great deal of time on the systems. Once they compromise a cage, intruders may download their toolkits, search for sensitive documents, attempt to gather and crack systems passwords, read system mail, review configuration files, modify system logs, or attempt other malicious activities. Besides wasting the attacker's time and resources, these actions give organizations time to react [23].

Figures 2.19 and 2.20 show two different screens of ManTrap.

2.15.9 Homemade Honeypots

Using a variety of commonly found security tools, some basic code, and a lot of creativity, one can create many different honeypots. There is no blueprint for developing ones own honeypot. It all depends on what one want it to do, the resources one have on hand, and the technologies one feel most comfortable with.

The variety of homemade honeypots is limited only by the imagination of security professionals. Two specific implementations of homemade honeypots are port monitoring and caged environments. These two types of honeypots represent the different extremes, from the low-interaction system to the more advanced caged environments.

Port monitoring is the simpler of homemade honeypots. Port monitoring honeypots are nothing more than a solution that monitors a specific port or a variety of ports. The goal of the port monitoring can be as simple as capturing connections to a service, such as with BOF, or it can entail response or emulation capabilities, such as with Specter. Either way, these solutions tend to be low interaction, limiting the attacker to

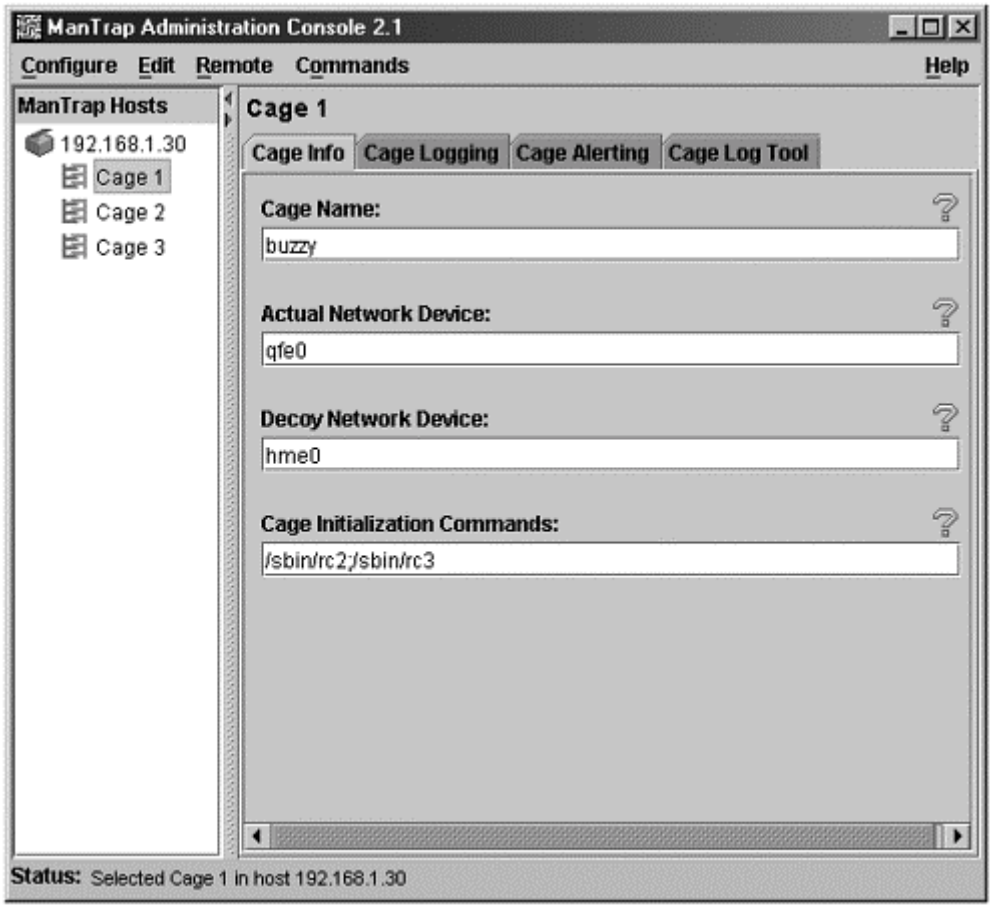

Figure 2.19: ManTrap GUI interface used to configure logging and alerting for each cage

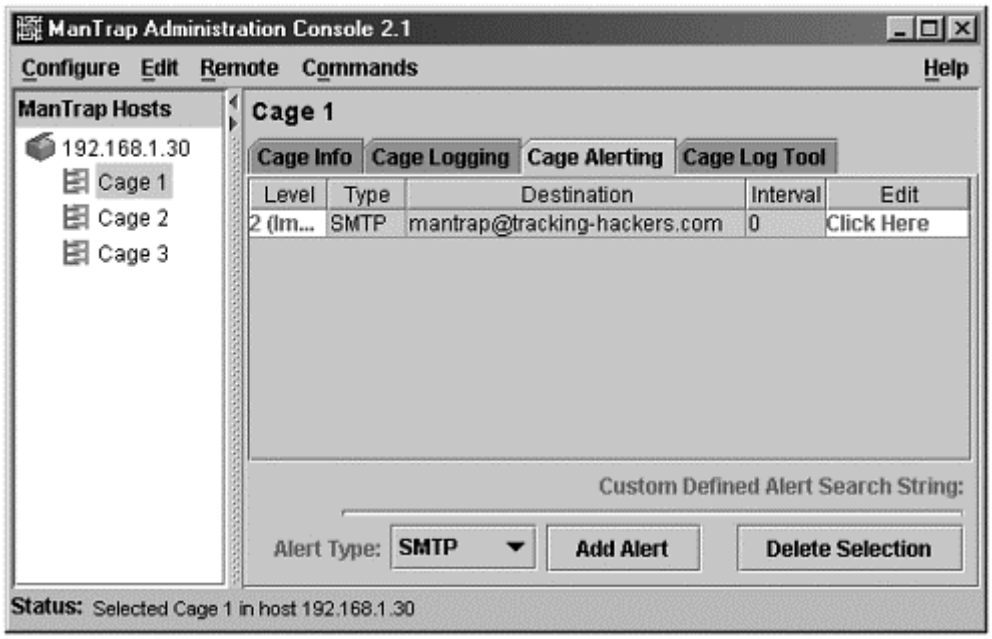

Figure 2.20: ManTrap GUI configured to set up alerts to `mantrap@tracking-hackers.com`

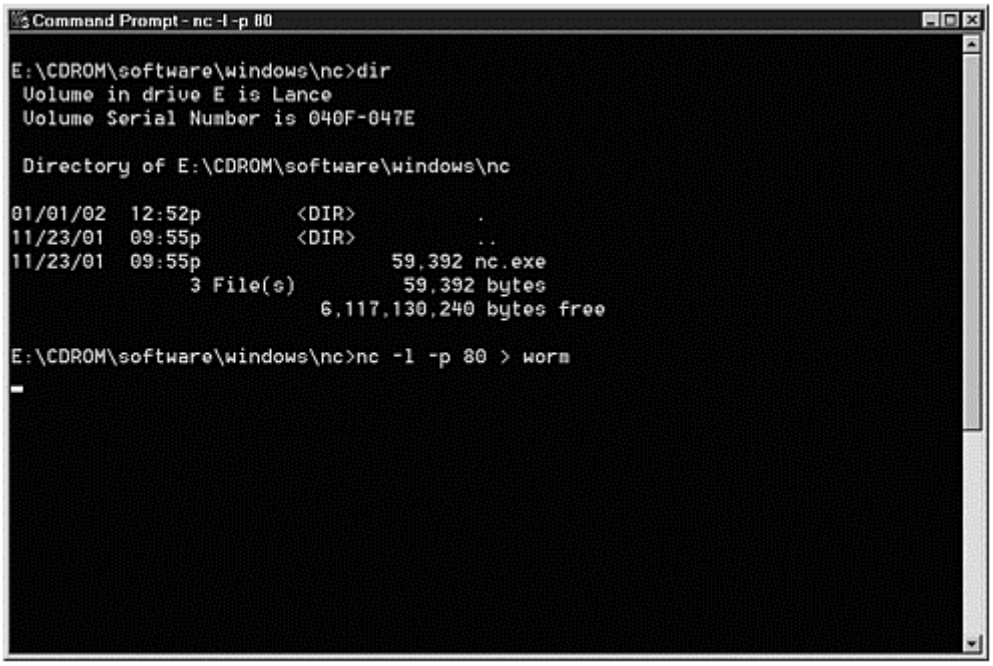

Figure 2.21: Running netcat on a Windows sytem to create a simple port listener on port 80

an emulated service to interact with.

A chroot or jailed environment is the more advanced technology. Instead of emulating the services, a caged environment is created. This caged environment exists within a real operating system. The advantage of the caged, or jailed, environment is that it creates the illusion of a real operating system. The attacker has nearly the same functionality as it would if it had compromised a real computer. However, its actions are more closely monitored and controlled [23].

A simple homemade honeypot and its working is shown in Figures 2.21,2.22 and 2.23.

2.15.10 Honeynets

Honeynets are high-interaction honeypots. It is difficult to conceive of a honeypot solution that can offer a greater level of interaction. The concept of a Honeynet is simple: Build a network of standard production systems, just as you would find in most organizations today. Put these network of systems behind some type of access control device (such as a firewall) and watch what happens. Attackers can probe, attack, and exploit any system within the Honeynet, giving them full operating systems and applications to interact with. No services are emulated, and no caged environments are created. The systems within a Honeynet can be anything: a Solaris server running an Oracle database, a Windows XP server running an IIS Web server, a Cisco router. In short, the systems within a Honeynet are true production systems.

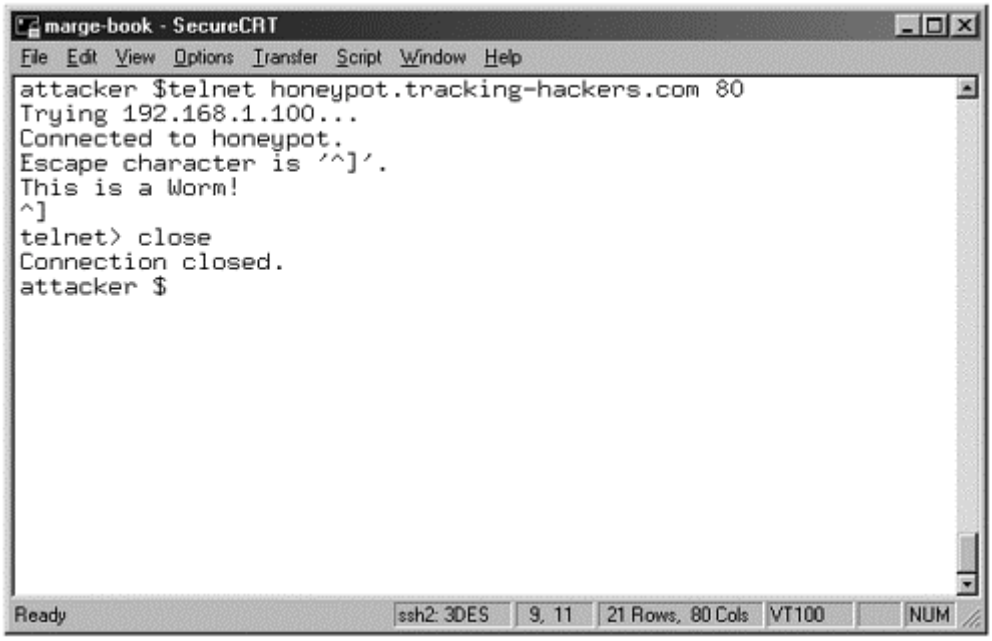

Figure 2.22: Attacker connecting to the port listener honeypot on port TCP 80

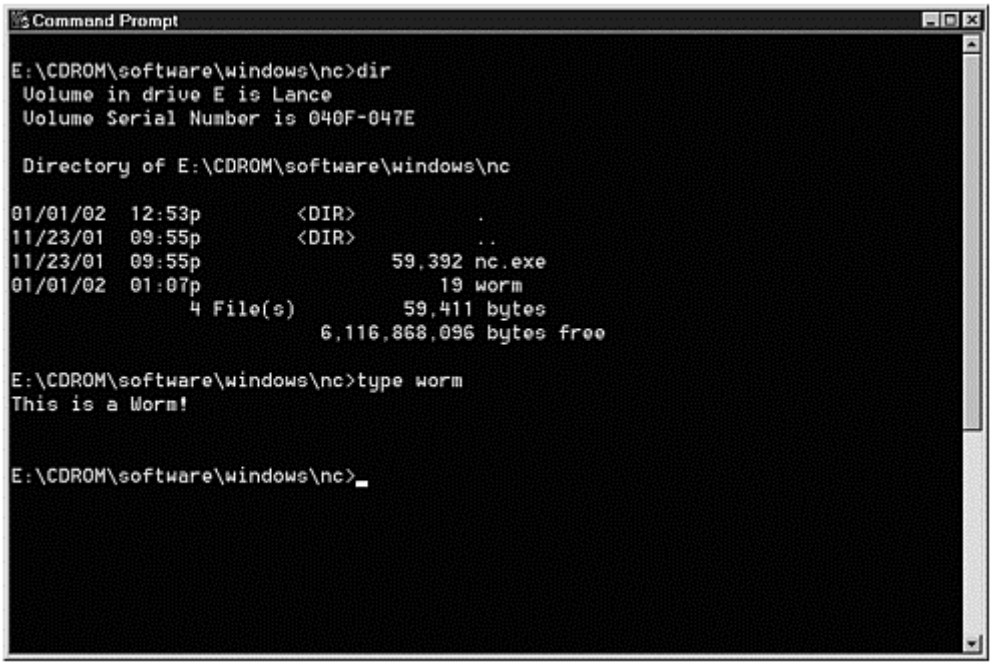

Figure 2.23: Contents of the file after the attacker completes the connection, netcat closes and saves the input to a file

A honeynet is essentially a research honeypot; its purpose is to collect information on attackers. It does this by creating a network of systems to be attacked. Nothing is emulated – it uses real systems and applications. The intent is for attackers to break into the system inside the honeynet and have their every action captured and controlled without them knowing it.

The challenge in building and deploying a honeynet is the lack of a prepackaged solution. One can't install a piece of software and have a honeynet ready to go. Instead, a honeynet is an architecture. In many ways it resembles a fishbowl, it's a contained environment in which one can watch everything happening. Once the architecture is built and the requirements are operational, target systems in the controlled network are deployed. Like the fishbowl, honeynet environment can contain whatever systems and have whatever appearance one like.

A honeynet architecture has two critical requirements: data control and data capture. One can use whatever technologies one want to enforce these requirements, but he must ensure that they are met. Data controls purpose is to reduce risk. Specifically, it ensures that once an attacker breaks into honeynets systems, those compromised systems cannot be used to attack or harm other systems. Data capture ensures that one can detect and capture all the attackers activities, even if they are obfuscated or encrypted.

2.15.10.1 Honeynet Data Control

Letting hackers break into honeypot systems but keeping them from breaking back out and harming other systems means target systems must be isolated in the honeynet with a layer-two bridging device. This forces all traffic going to and from the honeynet systems to first flow through an "invisible" layer-two bridge . Because the bridge operates at layer two, there is no time-to-live (TTL) decrement, packet routing, or MAC addresses for the attacker to identify. This bridge lets the bad guys come in, but it controls what they can do on their way out.

The key to data control, therefore, is allowing out bound activity but removing the ability to harm. The Honeynet Project in Gen-II uses Snort-Inline, a modified version of Snort, which is an open-source IDS technology . Instead of blocking detected outbound attacks, it modify and disable them. Attackers launch their exploits, which travel the Internet and hit their intended targets, but Snort-Inline disables the attacks, which ultimately fail. The attackers see the failure, but can't figure out why it occurred. One can continue to monitor the attackers while reducing the risk of harming remote systems

2.15.10.2 Honeynet Data Capture

To capture data the Honeynet uses kernel modules to insert in target systems. These kernel modules capture all the attacker's activities, such as encrypted keystrokes or scp

(secure copy, a tool used to encrypt file transfers) uploads of toolkits. The information the kernel module collects cannot be stored locally on the honeypot because the attacker could detect, accidentally delete, or modify it. Instead, the information must be remotely collected on a secure system, without the attacker's knowledge.

One successful method is to take the collected information and dump it on the network. The IDS gateway sniffs and captures all network activity on the honeynet to include any packets generated by the kernel module, thus the layer-two bridge also acts as network sniffer, capturing and logging all the attacker's activity. The trick to this is dumping the data without the attacker realizing it. Attackers could easily sniff the wire and see their own activity in the packet payload. To counter this, the kernel module spoofs the packets as NetBIOS traffic coming from other systems. Both source and destination IP and MAC addresses are spoofed so as to appear to be coming from a local Windows server. Even if the attacker sniffs the wire and sees the packets, they will seem to be normal traffic.

Figure 2.24 shows a First-generation Honeynet.The honeynet is nothing more than a contained environment. Positioned in this environment are the target systems (highlighted in yellow). Figure 2.25 shows a Second-generation (GenII) honeynet, a layer-two bridging device (called the honeynet sensor in the figure) isolates and contains systems in the honeynet.

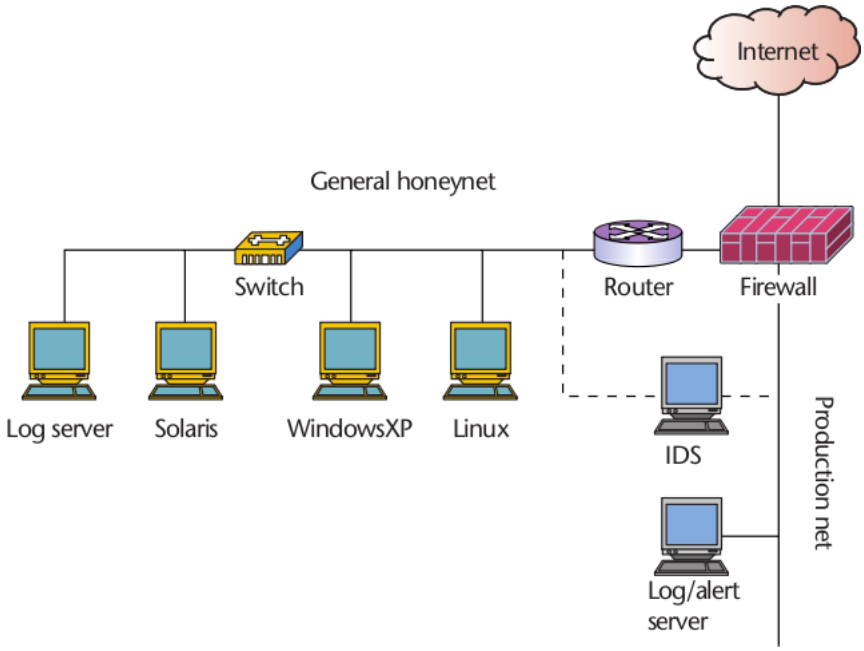

Figure 2.24: First-generation (GenI) honeynet

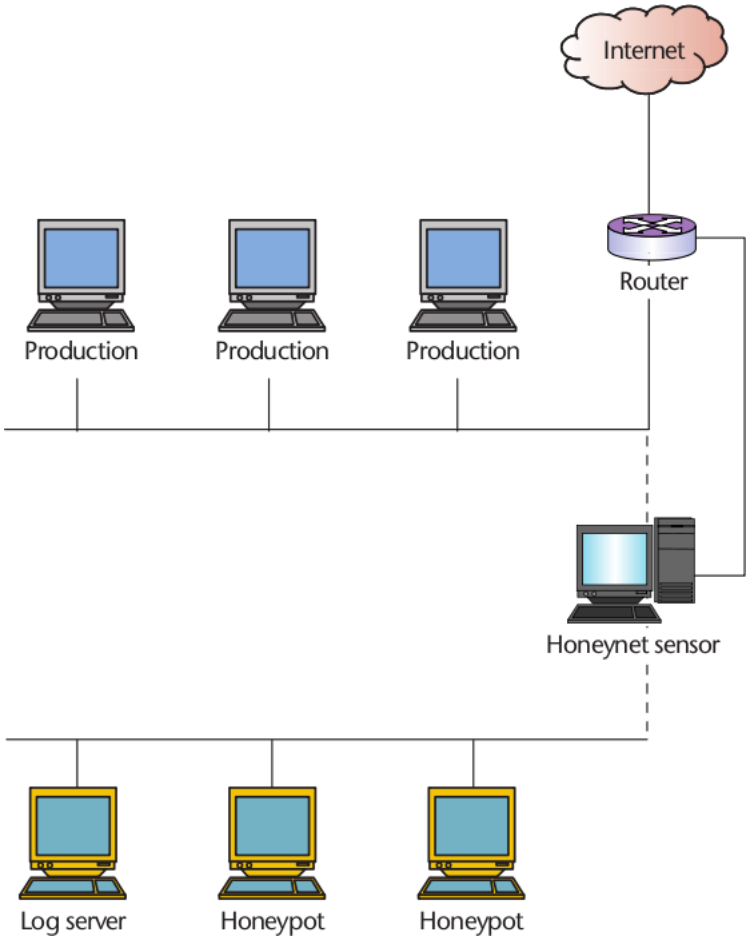

Figure 2.25: Second-generation (GenII) honeynet

Honeypot	**Interaction Level**	**OS Emulation**	**Free**	**Source**
HoneyBOT	Low	No	Yes	Closed
BackOfficer Friendly	Low	No	No	Closed
Specter	High	Yes	No	Closed
Honeyd	Low	Yes	Yes	Open
Deception Toolkit	Low	No	Yes	Open
Cybercop Sting	Low	Yes	No	Closed
NetFacade	Low	Yes	No	Closed
ManTrap	High	Yes	No	Closed
Homemade	Depends	Depends	Yes	Open
Honeynets	High	Yes	Yes	Open

Table 2.2: Comparison of Various Honeypots

2.15.11 Comparison of Various Honeypots

2.16 Related Technologies

2.16.1 Sebek

The Honeynet Project's Sebek tool addresses the challenge of host-based monitoring and capturing encrypted data. It employs a loadable kernel module approach (more often seen in blackhat rootkits) to handle host data capture in the kernel rather than the user space, making it much harder to detect and defeat. Trojaned read() calls capture all user keystrokes and file/socket read activity, which is then written to non-local storage via UDP network packets. This traffic is hidden fom system users by bypassing the host's own IP stack and instead writing the data directly to the network card device driver, effectively rendering it invisible to an attacker, even if root access and a network packet sniffer are available. The Sebek client is available for Linux, *BSD, Solaris and Windows operating systems.

Sebek is a data capture tool. As with all data capture tools, the goal is to capture data that will allow to accurately recreate the events on a honeypot. It enable us to determine information such as when an intruder broke in, how they did it, and what they did after gaining access. This information can, potentially, tell who the intruder is, what their motivations are, and who they may be working with. To determine what an intruder did after gaining access, data that provides the intruder's keystrokes and the impact of the attack is required.

When encryption is not used, it is possible to monitor the keystrokes of an intruder by capturing the network activity off of the wire and then using a tool like ethereal to reassemble the TCP flow and examine the contents of the session. When the session is encrypted, stream reassembly yields the encrypted contents of the session. To be of use these must be decrypted. This route has proven quite difficult for many.

Information that is encrypted must at some point be decrypted for it to be of any use. The process of circumvention involves capturing the data post decryption.The first versions of Sebek were designed to collect keystroke data from directly within the kernel. These early versions were the equivalent of a souped up Adore Rootkit that used a trojaned *sys_read* call to capture keystrokes. This system logged keystrokes to a hidden file and exported them over the network in a manner to make them look like other UDP traffic, such as NetBIOS. Current iteration of Sebek, version 2, was designed not only to record keystrokes but all *sys_read* data.

Sebek has two components: a client and server. The client captures data off of a honeypot and exports it to the network where is collected by the server (Figure 2.26). The server collects the data from one of two possible sources: the first is a live packet

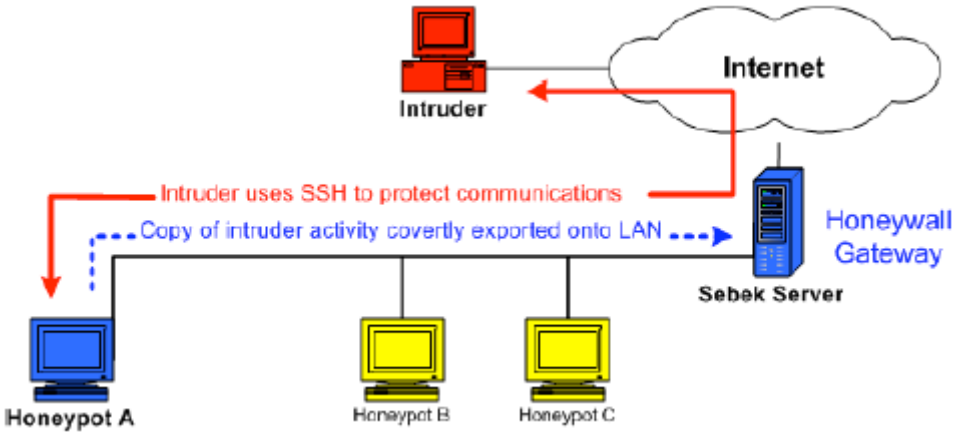

Figure 2.26: Typical Sebek deployment
[36]

capture from the network, the second is a packet capture archive stored as a tcpdump formatted file. Once the data is collected it is either uploaded into a relational database or the keystroke logs are immediately extracted. The communications used by Sebek are UDP based and as such are connectionless and unreliable [36].

Figure 2.26 shows a typical Sebek deployment. The client module is installed on the honeypot (in blue). The attacker's activity captured by the honeypot is dumped to the wire (hidden to the attacker) and passively collected by the Honeywall Gateway.

2.16.2 La Brea Tarpit

LaBrea is a program that creates a tarpit or, as some have called it, a "sticky honeypot". LaBrea takes over unused IP addresses on a network and creates "virtual machines" that answer to connection attempts. LaBrea answers those connection attempts in a way that causes the machine at the other end to get "stuck", sometimes for a very long time. The forged replies sent by the honeypot are crafted to maintain an open connection with the attacker, slowing down or even stopping the automated attack.

LaBrea works by watching ARP requests and replies. When the program sees consecutive ARP requests spaced several seconds apart, without any intervening ARP reply, it assumes that the IP in question is unoccupied. It then "creates" an ARP reply with a bogus MAC address, and fires it back to the requester [37].

2.16.3 Honeytokens

Honeytokens represent one of the newest and most interesting implementations of a honeypot. First, they are not a computer; instead they are a digital entity, such as an Excel spreadsheet. Even though they are not a computer, they share the same definition and concept of honeypots, no one should be interacting with them. Any interaction with

a honeytoken implies unauthorized or malicious activity. Second, they are extremely flexible, they have the ability to adapt to any environment. The reason for this is simple, a honeytoken can pretty much be anything you want. Examples can include a Word document, login and password, database record, or social security number [20].

Honeytoken is a honeypot that is not a computer. Instead it is some type of digital entity. A honeytoken can be a credit card number, Excel spreadsheet, PowerPoint presentation, a database entry, or even a bogus login. Honeytokens come in many shapes or sizes, however they all share the same concept: a digital or information system resource whose value lies in the unauthorized use of that resource. Just as a honeypot computer has no authorized value, no honeytoken has any authorized use. No one should be using or accessing a honeytoken. This gives honeytokens the same power and advantages as traditional honeypots, but extend their capabilities beyond just physical computers [38].

2.16.4 Honeyfarms

Honeyfarms is a groups of honeypots. Honeyfarms tend to be more centralized. Grouping honeypots provide many synergies that help to mitigate many of the deficiencies of traditional honeypots. For instance, honeypots often restrict outbound traffic in order to avoid attacking non-honeypot nodes. However, this restriction allows honeypots to be identified by an attacker. Honeyfarms can be used as redirection points for outbound traffic from each individual honeypot (Figure 2.27). These redirection nodes also behave like real victims . Following figure shows the redirection of outbound traffic from a honeypot to another node in the honeyfarm [21].

2.16.5 Distributed Honeypots

One disadvantage of honeypots is that must take up a large portion of the address space in order to be efficient and useful (since attackers and malware must target the honeypots). Distributed honeypots provide a distributed framework for grid computing in which legitimate hosts redirect suspicious users to a single honeypot. The client traffic is redirected anonymously through the Tor network to a collection of central honeypots [21].

2.16.6 Shadow Honeypot

Shadow honeypots are combination of honeypots and anomaly detection systems (ADS), which are another alternative to rule-based intrusion detection systems.

Shadow honeypots first segment anomalous traffic from regular traffic. The anomalous traffic is sent to a shadow honeypot which is an instance of a legitimate service as shown in Figure 2.28. If an attack is detected by the shadow honeypot, any changes

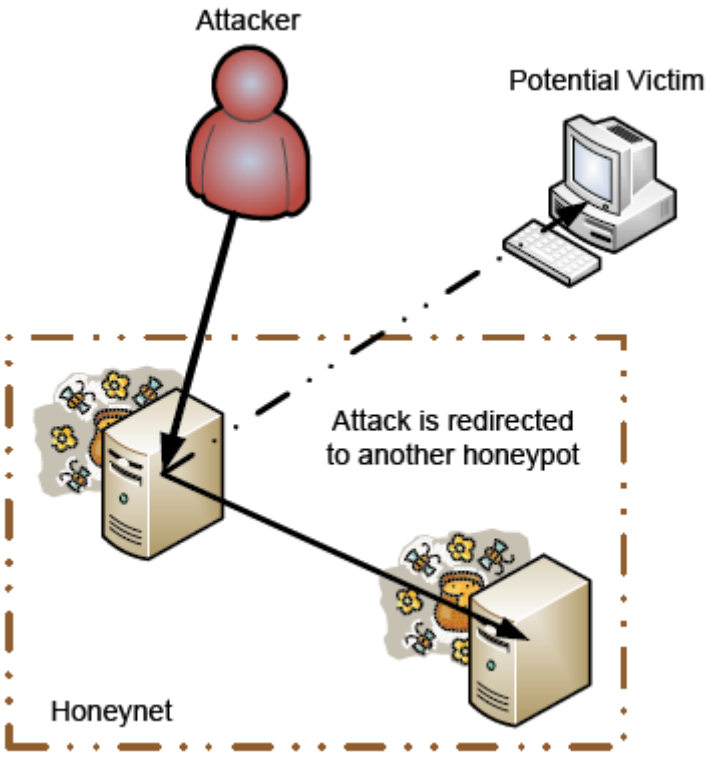

Figure 2.27: Redirecting an outbound attack in a honeynet
[21]

in state in the honeypot are discarded. If not, the transaction and changes are correctly handled. While shadow honeypots require more overhead, they are advantageous in that they can detect attacks contingent upon the state of the service [21].

2.17 Honeywall

Honeynets are nothing more than an architecture. To succesfully deploy a honeynet, one must correctly deploy the honeynet architecture. The key to the honeynet architecture is a honeywall. This is a gateway device that seperates honeypots from the rest of the world. Any traffic going to or from the honeypots must go through the honeywall. This gateway is traditionally a layer 2 bridging device, meaning the device should be invisible to anyone interacting with the honeypots. A diagram of this architecture is given below. In the figure the honeywall has 3 interfaces. The first 2 interfaces (eth0 and eth1) are what seperate these honeypots from everything else, these are bridged interfaces that have no IP stack. The 3rd interface (eth2, which is optional) has an IP stack allowing for remote administration. Figure 2.29 shows a Honeywall in a network.

There are several key requirements that a honeywall must implement namely, Data

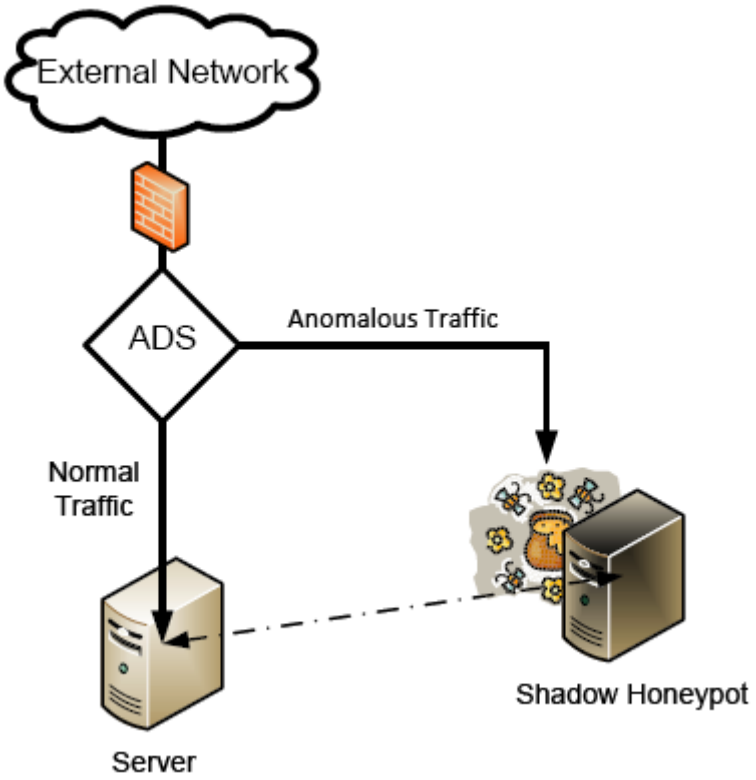

Figure 2.28: Segmenting traffic in a shadow honeypot system [21]

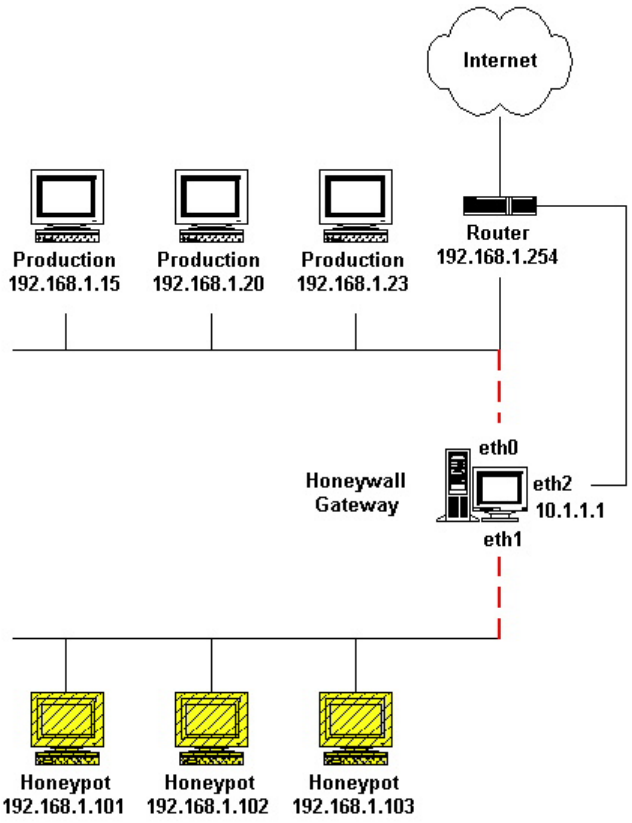

Figure 2.29: Honeywall in a Network

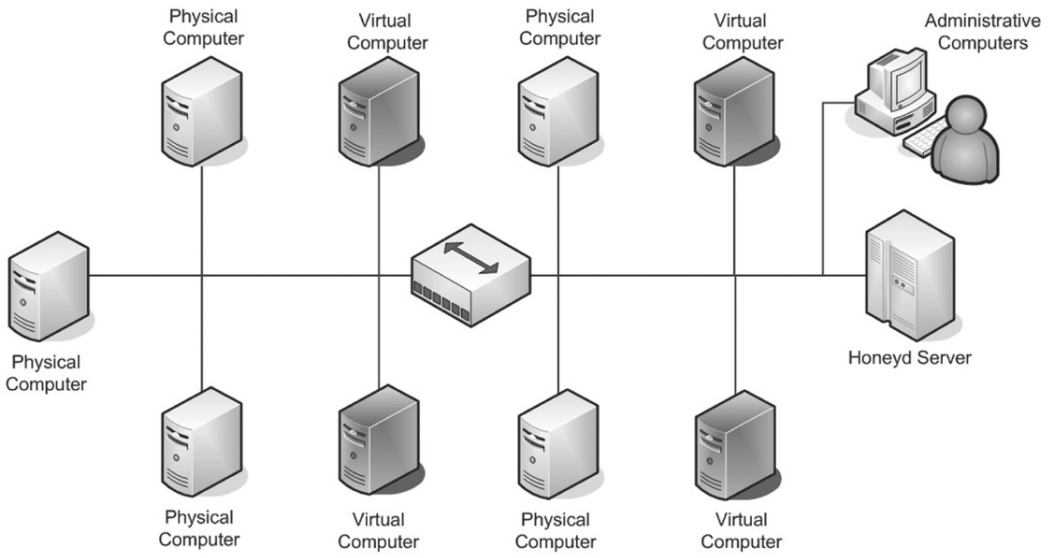

Figure 2.30: A Honeyd Sample Deployment

Control, Data Capture, Data Analysis and Data Collection. Data Control defines how activity is contained with the honeynet without an attacker knowing it. Its purpose is to minimize risk. Data Capture is capturing all of the attacker's activity without the attacker knowing it. Data Analysis is the ability to analyze this data. Data Collection is the ability to collect data from multiple honeynets to a single source. Of all these requirements, Data Control is the more important. Data Control always takes priority as its role is to mitigate risk [39].

2.18 Honeyd in Detail

Honeyd is described in detail in this section as it is being used by in this thesis to implement honeypot solution.

Honeyd is a framework to instrument thousands of Internet addresses with virtual honeypots and corresponding network services. Usually, Honeyd is configured to instrument-unallocated IP addresses on an existing network. For each IP address, Honeyd is told as the simulated computers are to behave. For example, one could set up a virtual web server that seems to run Linux and listens on port 80. One can create a virtual honeypot on another IP address with a network stack that looks like Windows on which all TCP ports seem to be running services. This would allow researchers to receive the first TCP payloads for worms or probes. Honeyd can be used to set up a few decoys in an existing network or to create routing topologies consisting of hundreds of networks and thousands of hosts with just a single computer.

It is a low-interaction virtual honeypot framework that can create thousands of virtual honeypots on a single network (Figure 2.30) or even all over the Internet. Honeyd

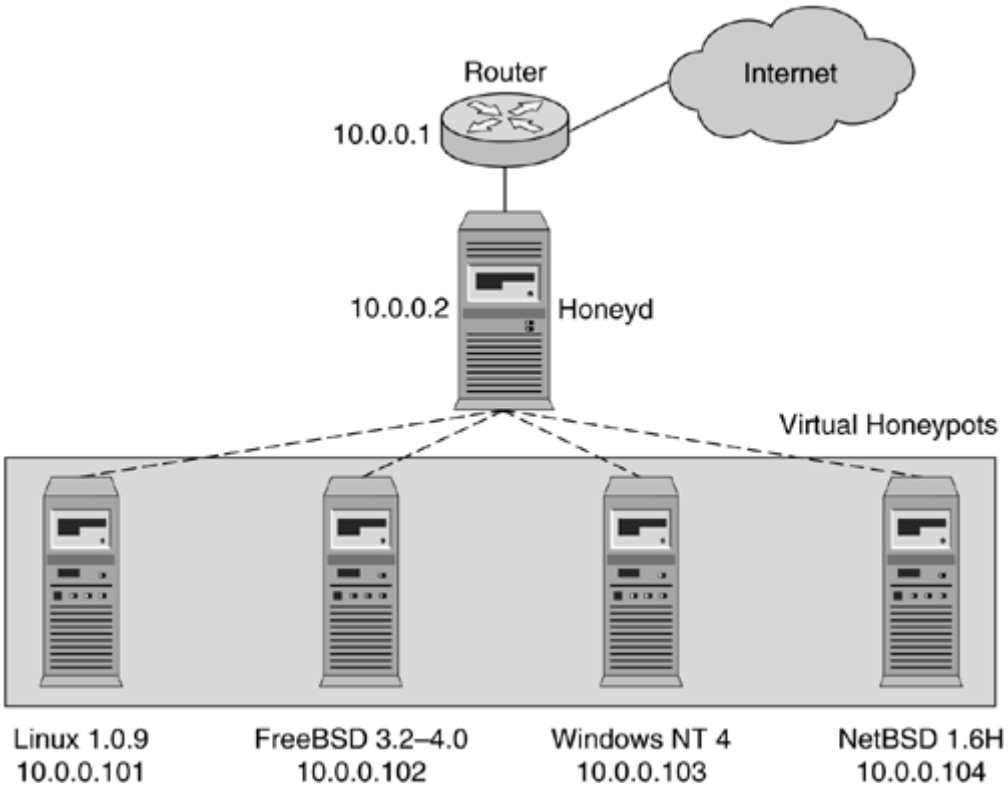

Figure 2.31: Honeyd Receiving Network Traffic

supports the IP protocol suites and responds to network requests for its virtual honeypots according to the services that are configured for each virtual honeypot. When sending a response packet, Honeyd's personality engine makes it match the network behavior of the configured operating system personality. It is available as an open source software released under the GNU Public License (GPL) and runs on most operating systems.

As shown in Figure 2.31, Honeyd receives traffic for its virtual honeypots via a router or Proxy ARP. For each honeypot, Honeyd can simulate the network stack behavior of a different operating system.

2.18.1 Honeyd Features

Honeyd has many interesting features:

- Simulates thousands of virtual hosts at the same time: The main reason for using Honeyd is its ability to create thousands of virtual honeypots at the same time. An adversary can interact with every single host via the network and experience different behavior from each host depending on how it has been configured.

- Configuration of arbitrary services via configuration file: One can provide arbitrary programs that interact with an adversary. Whenever Honeyd receives a new network connection, it will start the program that is specified for this connection

to talk back to the attacker. Instead of running programs, one can also use Honeyd to proxy connections to other machines or use features like passive fingerprinting to identify remote hosts and random sampling for load scaling.

- Simulates operating systems at TCP/IP stack level: This feature allows Honeyd to deceive Nmap and Xprobe into believing a virtual honeypot is running any configured operating system. To further increase realism, the policies for treating fragment reassembly and FIN scanning can be adjusted as well.

- Simulation of arbitrary routing topologies: The routing topologies can be arbitrarily complex. It is possible to configure latency, packet loss, and bandwidth characteristics. Honeyd supports asymetric routing, integration of physical machines into a virtual topology, and distributed operations via GRE tunnels.

- Subsystem virtualization: With subsystems, Honeyd can execute real Unix applications under the virtual name space of a honeypot, for example, web servers, ftp servers, and so on. This feature also allows for dynamic port binding in the virtual address space and background initiation of network connections.

2.18.2 Honeyd Architecture

Honeyd employs a simple architecture (Figure 2.32). A central packet dispatcher receives all interesting network traffic. Based on the specific configuration, different service processes are created to handle the traffic. Every packet that is being sent back to the network is modified by a personality engine to match the characteristics of the configured operating system.

2.18.3 Receiving Network Data

A central machine intercepts network traffic sent to the IP addresses of configured honeypots and simulates their responses. Before any packets are received, network needs to be configured so that packets for the virtual honeypots reach the Honeyd host. Honeyd replies to network packets whose destination IP address belongs to one of the simulated honeypots. However, lacking the proper network configuration, Honeyd is never going to see any packets to reply to. The network needs to be configured in such a fashion that Honeyd receives packets for the IP addresses it simulates. There are several ways to do this. For example, one can create special routes for the virtual IP addresses that point to the Honeyd host, can use Proxy ARP, or can use network tunnels.

For example in Figure 2.33, a host on the internal network reaches the virtual honeypot directly via the local network. It needs to map the IP address to a MAC address via an ARP request and then send packets to the virtual honeypot. An external host does

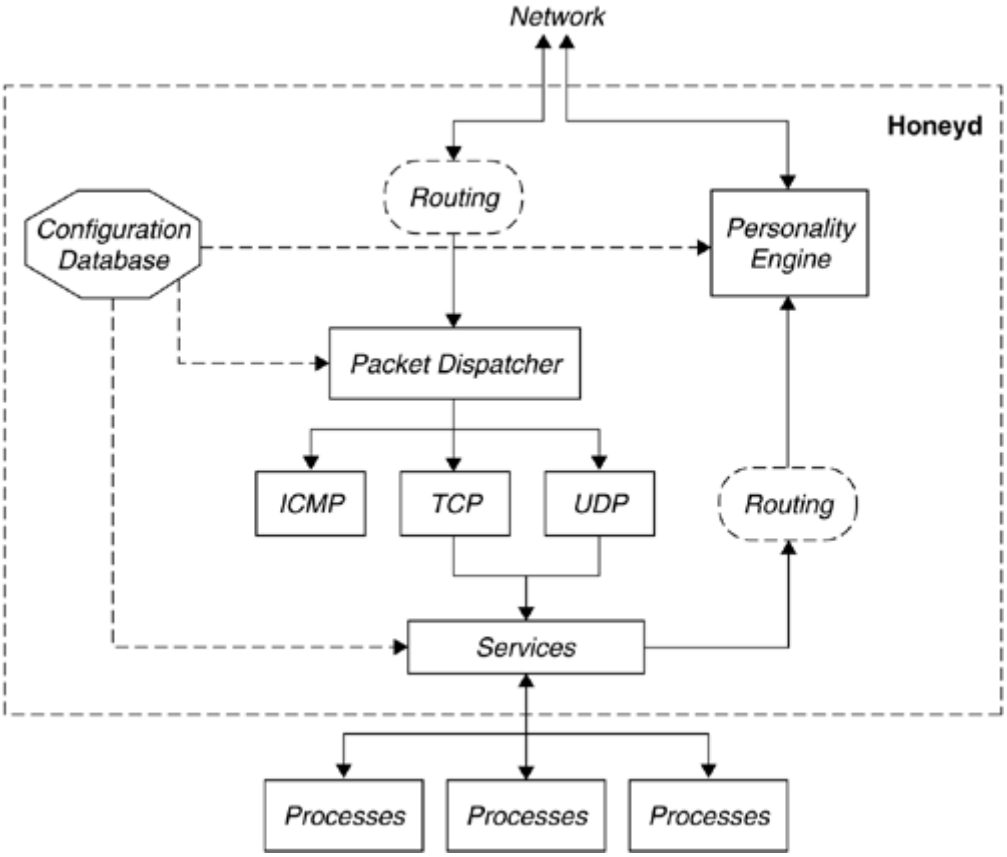

Figure 2.32: Honeyd Architecture

not need to know about MAC addresses because the router is going to send the ARP request.

2.18.4 Configuration

Honeyd uses a simple text-based configuration file to specify on which IP addresses to run virtual honeypots and also to specify which services are available for each host. The configuration language is a context-free-grammar that can be described in BNF (Backus-Naur form). The BNF specification of the Honeyd's configuration language is shown below:

```
config = creation | addition | delete | binding | set |
       annotate | route [config] | option
creation= "create" template-name | "create" "default" |
  "dynamic" template-name
addition= "add" template-name proto "port" port-number action |
  "add" template-name "subsystem" cmd-string ["shared"] ["restart"] |
  "add" template-name "use" template-name "if" condition
delete= "delete" template-name |
```

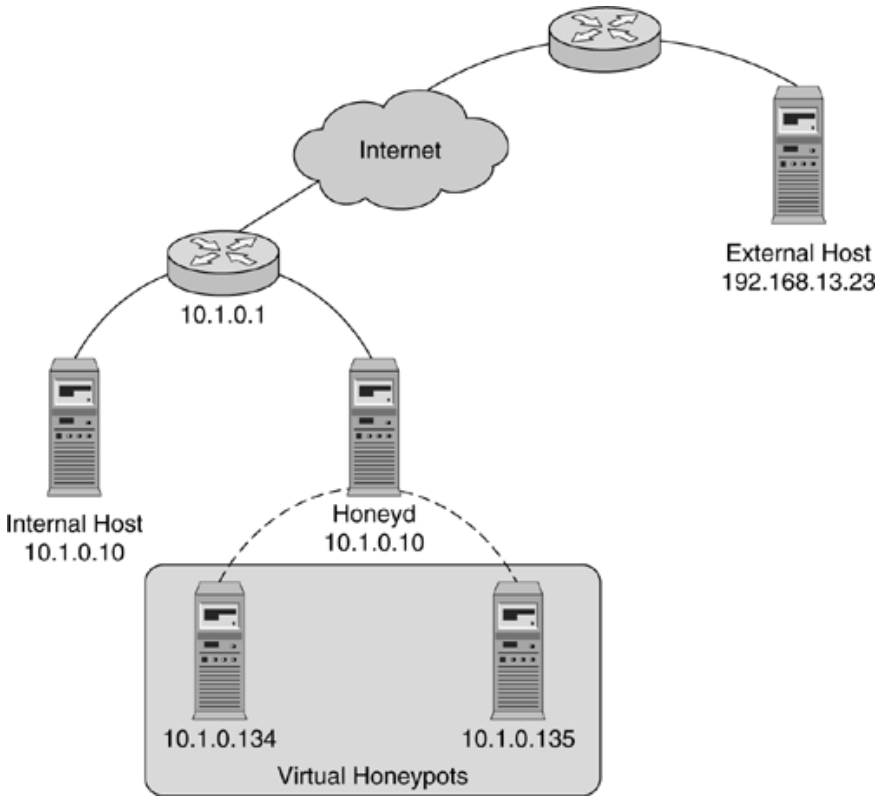

Figure 2.33: Honeyd Receiving Network Data

```
  "delete" template-name proto "port" port-number
binding = "bind" ip-address template-name |
  "bind" condition ip-address template-name |
  "bind" ip-address "to" interface-name |
  "dhcp" template-name "on" interface-name ["ethernet" cmd-string] |
  "clone" template-name template-name
set = "set" template-name "default" proto "action" action |
  "set" template-name "personality" personality-name |
  "set" template-name "personality" "random" |
  "set" template-name "ethernet" cmd-string |
  "set" template-name "uptime" seconds |
  "set" template-name "droprate" "in" percent |
  "set" <template-name> "maxfds" <number> |
  "set" template-name "uid" number ["gid" number] |
  "set" ip-address "uptime" seconds
annotate= "annotate" personality-name [no] finscan |
  "annotate" personality-name "fragment" ("drop" | "old" | "new")
route = "route" "entry" ipaddr |
  "route" "entry" ipaddr "network" ipnetwork |
  "route" ipaddr "link" ipnetwork |
```

```
  "route" ipaddr "unreach" ipnetwork |
  "route" ipaddr "add" "net" ipnetwork \\
             "tunnel" ipaddr(src) ipaddr(dst) |
  "route" ipaddr "add" "net" ipnetwork ipaddr \\
             ["latency" number"ms"] ["loss" percent] \\
             ["bandwidth" number["Mbps"|"Kbps"] \\
             ["drop" "between" number "ms" "-" number "ms" ]
proto = "tcp" | "udp" | "icmp"
action = ["tarpit"] ("block" | "open" | "reset" | cmd-string | \\
  "internal" cmd-string \\
  "proxy" ipaddr":"port )
condition = "source os =" cmd-string |
  "source ip =" ipaddr | "source ip =" ipnetwork |
  "time " timecondition
```

An example configuration for Honeyd is shown below. The configuration language is a context-free grammar. This example defines two templates: a router that can be accessed via telnet and a host that is running a web server.

```
create routerone
set routerone personality "Cisco 7206 running IOS 11.1(24)"
set routerone default tcp action reset
add routerone tcp port 23 "scripts/router-telnet.pl"
create netbsd
set netbsd personality "NetBSD 1.5.2 running on a Commodore Amiga
(68040 processor)"
set netbsd default tcp action reset
add netbsd tcp port 22 proxy \$ipsrc:22
add netbsd tcp port 80 "scripts/web.sh"
bind 10.1.0.1 routerone
bind 10.1.0.134 netbsd
```

Chapter 3

Problem Statement

3.1 Problem Definition

It was desired to make a honeypot solution which could be installed in an academic campus. Although there are many honeypots available today, but not all of them can be used as campus honeypot because of the following reasons:

1. Advanced honeypots designed to be used in a corporate environment are either not free or even if they are free, they are overkilled for being used as a simple campus honeypot.

2. Simple honeypots does exist but these does not have any option to log their output in a way that can be viewed easily and remotely.

It was desired to make a honeypot which will be easy to install and deploy, it should be absolutely free and open-source. It should also provide capabilities to view logs and output from a remote location so that an administrator can monitor the status of honeypot from his/her home and need not to go to computer lab where honeypot is deployed to check the status of honeypot.

3.2 Objectives

1. To design and develop a honeypot solution for proactive monitoring of campus network.

2. To provide capabilities to remotely view the honeypot's status.

3. To enable easy access to data and make searching for required information easy.

Chapter 4

Implementation of Campus Honeypot

4.1 Honeyd as honeypot

Out of all existing honeypots, Honeyd is chosen because of the following features:

1. Honeyd is free and open-source.
2. Honeyd is highly configurable. Many different types of OS personalities can be specified in its configuration file.
3. Honeyd emulates different operating systems from TCP/IP stack level and hence can fool OS fingerprinting softwares likes nmap.

4.2 OS Personality

An OS personality is the virtual OS that is emulated by the Honeyd in order to fool OS fingerprinting softwares like nmap. Honeyd uses nmap's own fingerprint database for this purpose and modifies the outgoing's packets according to the network stack of the configured OS.

4.3 Nmap OS Detection

Nmap is the network exploration tool and security/port scanner. It has a feature to scan the type of OS running on a remote system. To invoke OS detection -O switch is required. OS detection is performed by sending different probe packets to the remote host and reading its response. Figure 4.1 shows a screenshot of nmap running and scanning the OS of a remote machine.

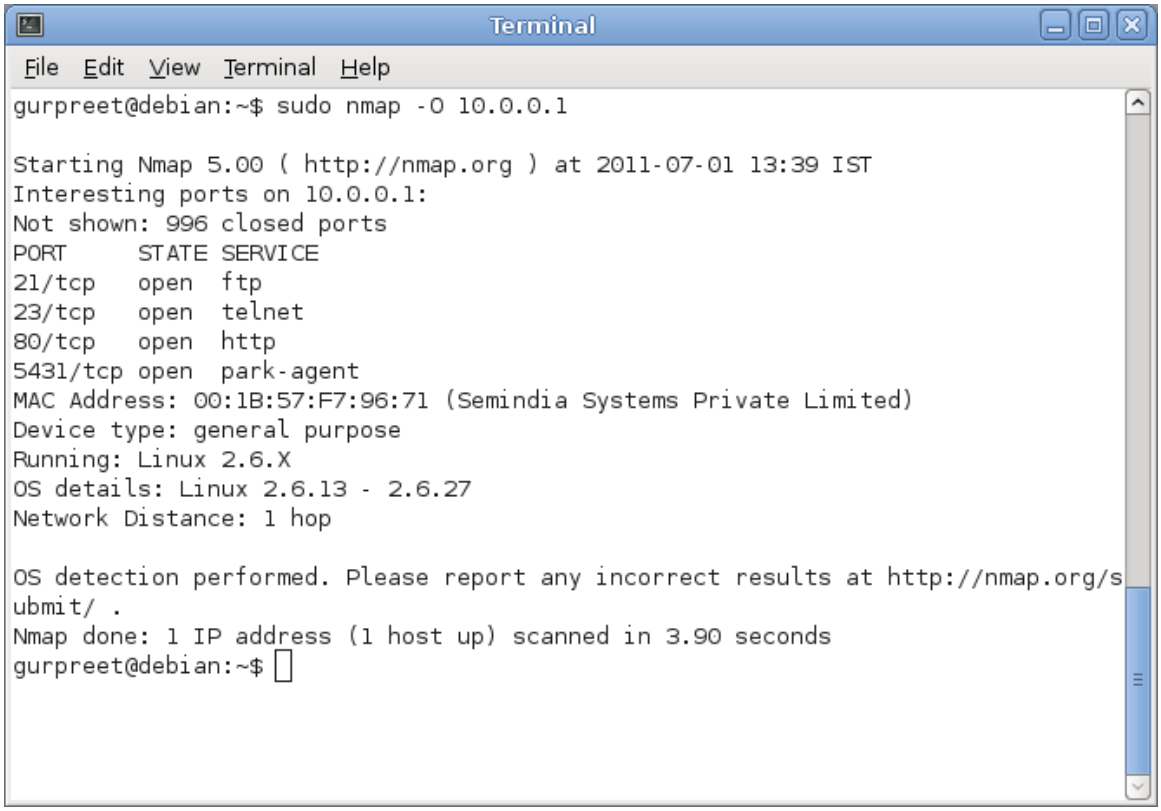

Figure 4.1: Nmap Detecting OS of Remote Machine

4.4 Nmap OS Fingerprints

In order to do OS detection, nmap perform several tests by sending different probe packets to the remote system and then matching its response to its OS fingerprint database. An example of an OS fingerprint is shown below:

```
# OpenSUSE 10.2 Linux linux 2.6.18.8-0.3-default #1 SMP Tue Apr 17 \
08:42:35 UTC 2007i686 athlon i386 GNU/Linux
Fingerprint Linux 2.6.18.8 (openSUSE 10.2, SMP)
Class Linux | Linux | 2.6.X | general purpose
SEQ(SP=C5-CF%GCD=1-6%ISR=C9-D3%TI=Z%II=I%TS=8)
OPS(O1=M5B4ST11NW4%O2=M5B4ST11NW4%O3=M5B4NNT11NW4%O4=M5B4ST11NW4%O5=\
M5B4ST11NW4%O6=M5B4ST11)
WIN(W1=16A0%W2=16A0%W3=16A0%W4=16A0%W5=16A0%W6=16A0)
ECN(R=Y%DF=Y%T=3B-45%TG=40%W=16D0%O=M5B4NNSNW4%CC=N%Q=)
T1(R=Y%DF=Y%T=3B-45%TG=40%S=O%A=S+%F=AS%RD=0%Q=)
T2(R=N)
T3(R=Y%DF=Y%T=3B-45%TG=40%W=16A0%S=O%A=S+%F=AS%O=M5B4ST11NW4%RD=0%Q=)
T4(R=Y%DF=Y%T=3B-45%TG=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T5(R=Y%DF=Y%T=3B-45%TG=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)
```

```
T6(R=Y%DF=Y%T=3B-45%TG=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T7(R=N)
U1(R=N)
IE(DFI=N%T=3B-45%TG=40%CD=S)
```

Here SEQ, OPS, WIN etc are various tests that nmap performs.

Honeyd tricks nmap's OS detection by reading the OS fingerprint of the desired personality and its personality engine modifies the outgoing packet to match the protocol stack of the configured operating system.

4.5 Version mismatch of OS Fingerprints

Nmap started using a new type of OS fingerprint since version 4.20, which is different from those used by Honeyd. Now as the two OS fingerprints of same OS does not match anymore. Honeyd could not reliably change the outgoing packet to make it match the network stack of configured OS. For example OS fingerprints of Nmap and Honeyd for the same OS are given below:

Nmap's Fingerprint

```
Fingerprint Apple Mac OS X Server 10.2.8 (Jaguar) (Darwin 6.8, PowerPC)
Class Apple | Mac OS X | 10.2.X | general purpose
SEQ(SP=FB-111%GCD=1-6%ISR=104-10E%TI=I%II=I%SS=S%TS=1)
OPS(O1=M5B4NW0NNT11%O2=M5B4NW0NNT11%O3=M5B4NW0NNT11%O4=M5B4NW0NNT11%O5=\
M5B4NW0NNT11%O6=M5B4NNT11)
WIN(W1=8218%W2=8220%W3=8204%W4=80E8%W5=80F4%W6=807A)
ECN(R=Y%DF=Y%T=3B-45%TG=40%W=832C%O=M5B4NW0%CC=N%Q=)
T1(R=Y%DF=Y%T=3B-45%TG=40%S=O%A=S+%F=AS%RD=0%Q=)
T2(R=N)
T3(R=Y%DF=Y%T=3B-45%TG=40%W=807A%S=O%A=S+%F=AS%O=M5B4NW0NNT11%RD=0%Q=)
T4(R=Y%DF=Y%T=3B-45%TG=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T5(R=Y%DF=N%T=3B-45%TG=40%W=0%S=Z%A=S+%F=AR%O=%RD=0%Q=)
T6(R=Y%DF=Y%T=3B-45%TG=40%W=0%S=A%A=Z%F=R%O=%RD=0%Q=)
T7(R=Y%DF=N%T=3B-45%TG=40%W=0%S=Z%A=S%F=AR%O=%RD=0%Q=)
U1(DF=N%T=3B-45%TG=40%IPL=38%UN=0%RIPL=G%RID=G%RIPCK=G%RUCK=0%RUD=G)
IE(DFI=S%T=3B-45%TG=40%CD=S)
```

Honeyd's Fingerprint

```
Fingerprint Apple Mac OS X 10.2.6 (Jaguar)
Class Apple | Mac OS X | 10.2.X | general purpose
TSeq(Class=TR%gcd=<6%IPID=I%TS=2HZ)
```

```
T1(DF=Y%W=209D%ACK=S++%Flags=AS%Ops=MNWNNT)
T2(Resp=N)
T3(Resp=Y%DF=Y%W=209D%ACK=S++%Flags=AS%Ops=MNWNNT)
T4(DF=N%W=0%ACK=O%Flags=R%Ops=)
T5(DF=N%W=0%ACK=S++%Flags=AR%Ops=)
T6(DF=N%W=0%ACK=O%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=0%ULEN=134%DAT=E)
```

One can easily see that the fingerprints are different, so Honeyd is unable to correctly modify outgoing packets.

4.6 Solution of Fingerprint Mismatch

An easy solution of the above shown fingerprint mismatch problem is to test all the personalities of Honeyd and see if there is any personality that when emulated makes nmap think that remote OS is running Linux or Windows. In other words it doesn't matter if personality is named Minix as long as it is making nmap think that Linux is running, or it doesn't matter if personality is named OS/2 as long as nmap is thinking it as Windows XP.

In order to test all personalities an script is developed to enable all the available personalities in the Honeyd's configuration file. The script automatically makes a large configuration file of honeyd which contains all the possible configurations with different IP addresses. The script and its related files are shown in Appendix A.

After running the honeyd with the automatically generated script and running nmap against each virtual honeypot thus created it was found that there were several personalities which fool nmap that remote OS is Linux and also there are several personalities which fool nmap that remote OS is running Windows.

- Personality for Linux – *"Linux 2.5.25 - 2.5.70 or Gentoo 1.2 Linux 2.4.19 rc1-rc7)"*
- Personality for Windows – *"Microsoft Windows Server 2003"*

4.7 Honeyd Configuration File

Below is the configuration file used for Honeyd, comments start with #.

```
#create a new template named 'template'
create template
```

```
#all the TCP ports should be blocked by default
set template default tcp action block

#all the UDP ports should be blocked by default
set template default udp action block

#ICMP should be enabled by default
set template default icmp action open

#open TCP ports 23 (telnet), 443 (HTTPS)  and 80 (HTTP)
add template tcp port 23 open
add template tcp port 443 open
add template tcp port 80 open

#personality "Microsoft Windows Server 2003" should use template\
# 'template'
set template personality "Microsoft Windows Server 2003"

#this template should be assigned IP 10.0.0.11
bind 10.0.0.11 template

#personality "Linux 2.5.25 - 2.5.70 or
#Gentoo 1.2 Linux 2.4.19 rc1-rc7)"
#should use template 'template'
set template personality "Linux 2.5.25 - 2.5.70 or \
Gentoo 1.2 Linux 2.4.19 rc1-rc7)"

#this template should be assigned IP 10.0.0.10
bind 10.0.0.10 template
```

Thus, when Honeyd will be run with above configuration file then two virtual honeypots will be created A Windows machine (IP 10.0.0.11) and Linux machine (IP 10.0.0.10), both will be having only TCP port 23,443 and 80 open and ICMP response enabled.

4.8 Arpd

Honeyd cannot route network packets not destined to it to itself. An external way is needed to do same. Three possible solutions exist:

1. Add a new ARP entry on the attacker's machine which point to machine running Honeyd. This is not possible due to obvious reason that one would not have access to the attacker's machine.

2. Add a new network route on the router. But this requires access to the router and permissions to add a new route.

3. Use Farpd program. Farpd program sends fake ARP replies for non-existing machines. Fake ARP replies contain MAC address of machine running Arpd and Honeyd.

Hence, Farpd program is used to send out fake ARP replies, original name of the program was arpd but it is renamed as farpd because of conflicts with another Debian package. The program name farpd has been changed in Debian GNU/Linux from the original name (arpd) to avoid name clash with other ARP daemons. Hence for the purpose of the thesis Farpd and Arpd refer to same program. From Farpd man page:

> *farpd replies to any ARP request for an IP address matching the specified destination net with the hardware MAC address of the specified interface, but only after determining if another host already claims it.*

However, the man page also tell about two bugs which directly affect the usage of Farpd in the honeypot setup:

> *farpd will respond too slowly to ARP requests for some applications. In order to ensure that it does not claim existing IP addresses it will send two ARP request and wait for a reply. This slowness affects the nmap net work scanning tool, and possibly others, which uses by default ARP when scanning local networks. The answers from farpd will come after the tool has timeout waiting for the ARP replies and, consequently, IP addresses claimed by farpd will not be discovered.*
>
> *Additionally, farpd sends the ARP replies to the broadcast address of the network and not to the host that send the ARP request. Some systems and applications (notably nmap) will not handled these requests and expect directed ARP replies (i.e. targeted specifically to the host that sent the request and not to the network)*

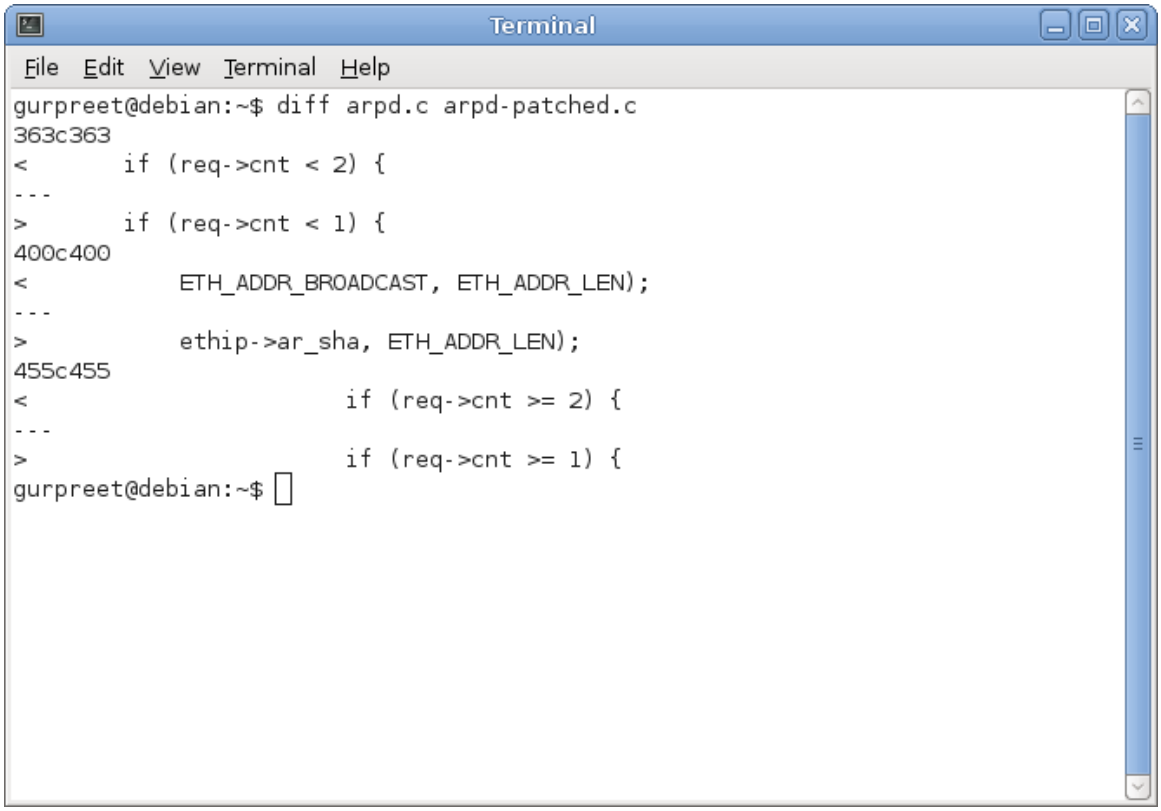

Figure 4.2: Diff of patch in farpd program's ard.c file

Hence, in order to correct these problems farpd is patched and installed. The difference of the patch is given in the Figure 4.2.

This small patch enabled arpd to send unicast ARP reply instead of broadcasting it and also allowed it to send ARP reply faster, as now it only waits for 1 time for the ARP reply instead of 2.

4.9 Starting Honeyd and Arpd

To use honeyd with arpd, both needs to be started separately in different terminals. The Honeyd is started (Figure 4.3) with the following command line:

"sudo honeyd -d -i eth0 -f research/config -l /tmp/honeydlog"

Decription of command line:

sudo to run honeyd as root

honeyd the name of the program

-d run in debug mode and in foreground, otherwise honeyd would have started in background

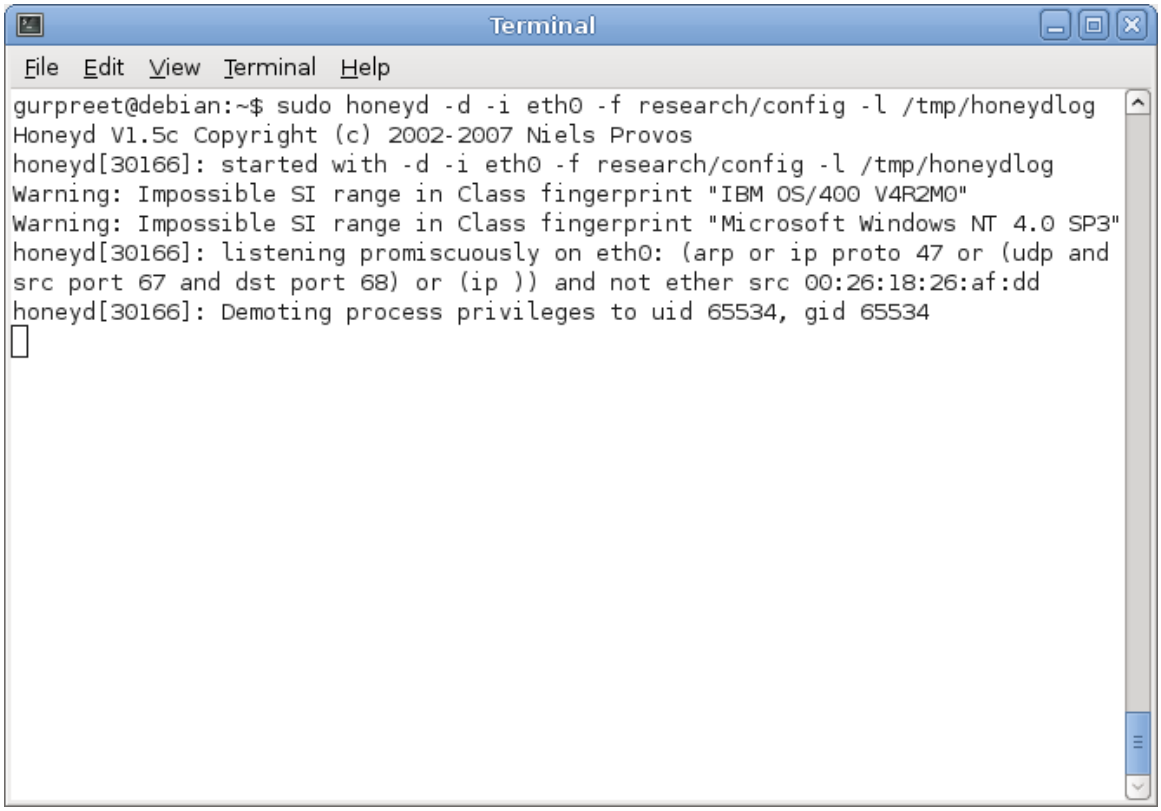

Figure 4.3: Screenshot of Honeyd starting

-i eth0 use eth0 interface

-f researcher/config location of the configuration file

-l /tmp/honeydlog generate a log file at the specified location

To start arpd (Figure 4.4), following command is used:

"sudo arpd -d -i eth0"

Decription of command line:

sudo to run arpd as root

arpd the name of the program

-d do not go to background

-i eth0 use eth0 interface

4.10 Raw Output of Honeyd and Arpd

Next output and response of Honeyd and arpd will be shown when an attack pings, telnets or scans virtual honeypots.

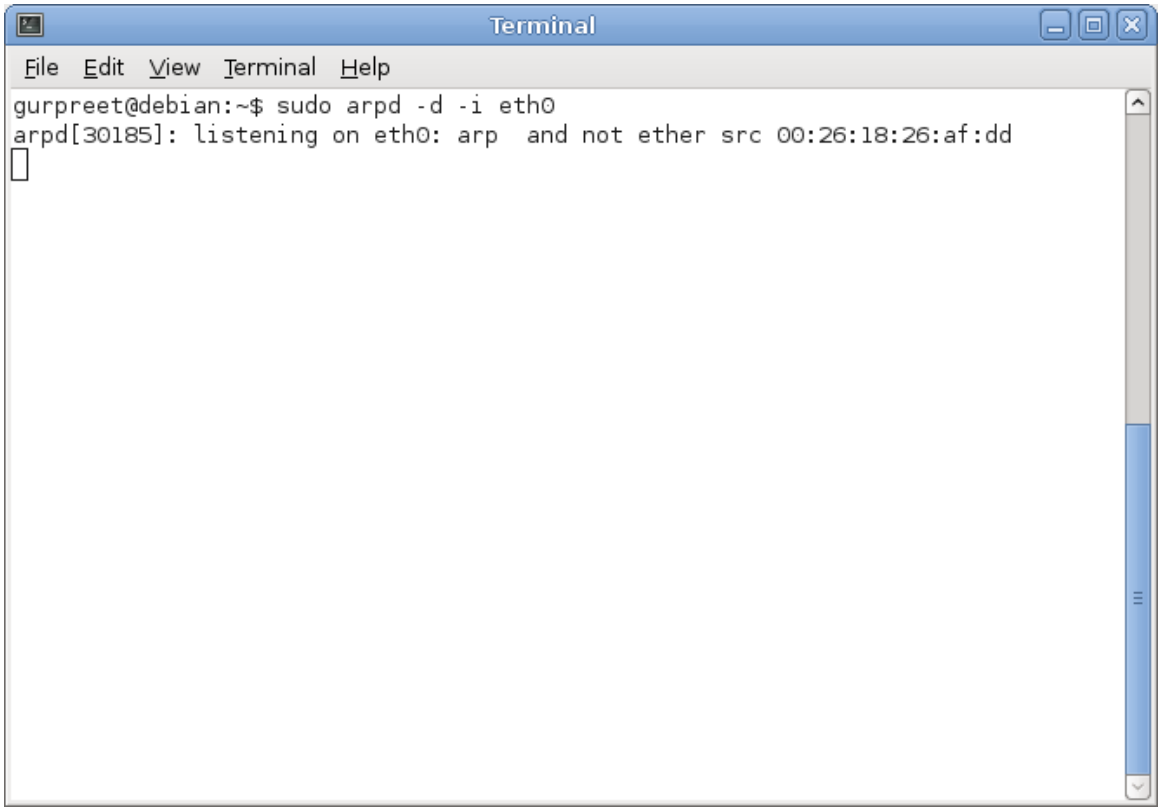

Figure 4.4: Screenshot of Arpd starting

4.10.1 Scenario 1 - Attacker pinging virtual honeypot

Assuming 10.0.0.3 is the attacker and he is trying to ping the virtual honeypot say 10.0.0.10 (i.e. the Linux machine).

Step 1 - Attacker would run the command:

ping -c 2 10.0.0.10

-c 2 to ping 2 times

When attacker will try to ping 10.0.0.10, first his system will try to find the MAC address of remote system i.e. system having IP 10.0.0.10. The arpd running on 10.0.0.2 will reply with the MAC address of 10.0.0.2 and so attacker's system will send ping packet destined for 10.0.0.10 to 10.0.0.2 instead. This will be more clear with output of tcpdump at both attacker and on victim side in Figures 4.5 and 4.6.

Step 2 - arpd responding

Arpd will respond to the ping as shown in Figure 4.7.

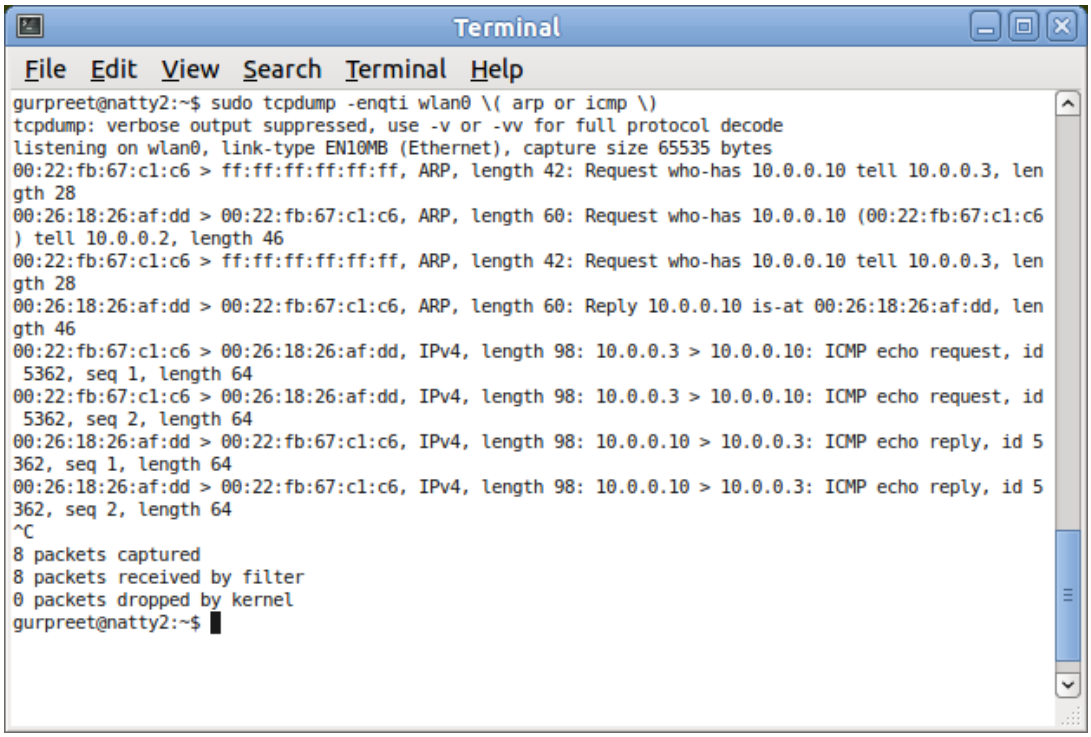

Figure 4.5: Tcpdump at attacker side

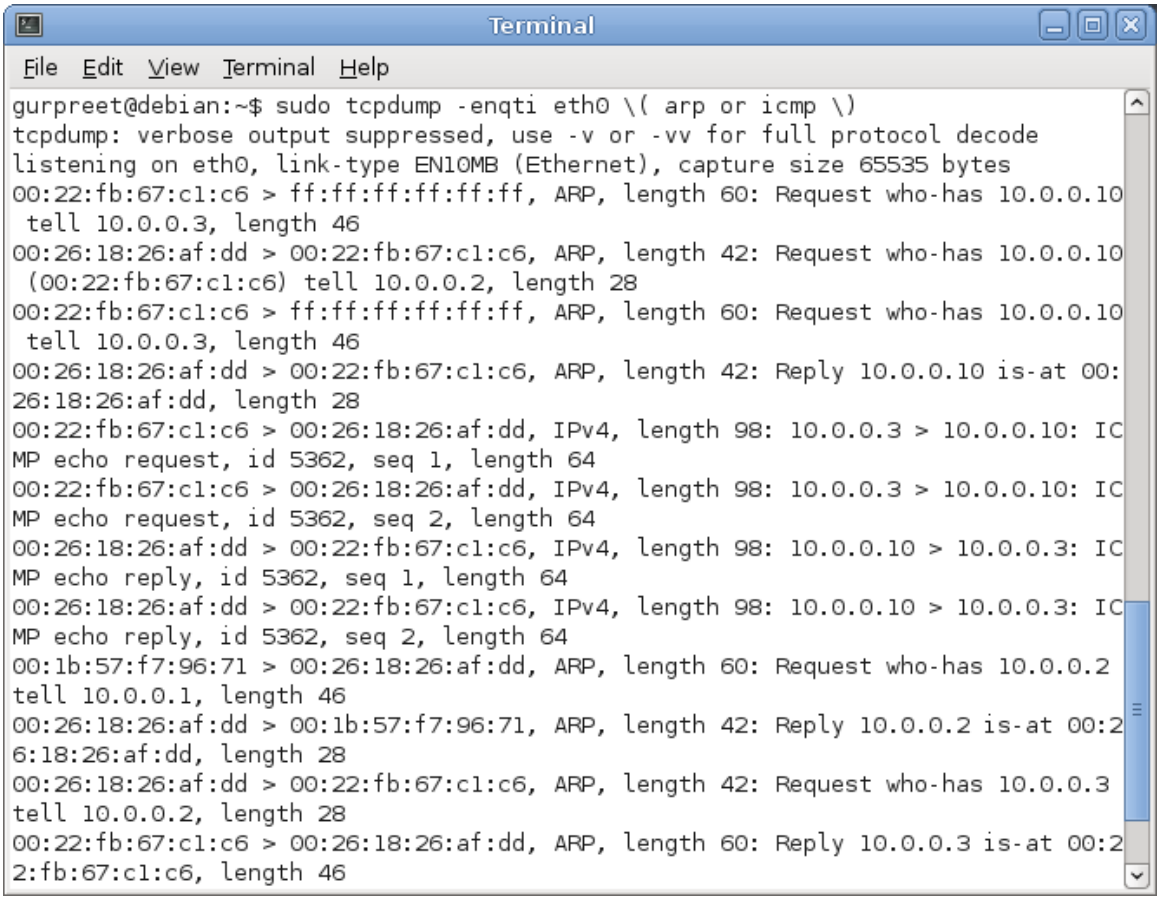

Figure 4.6: Tcpdump at victim side

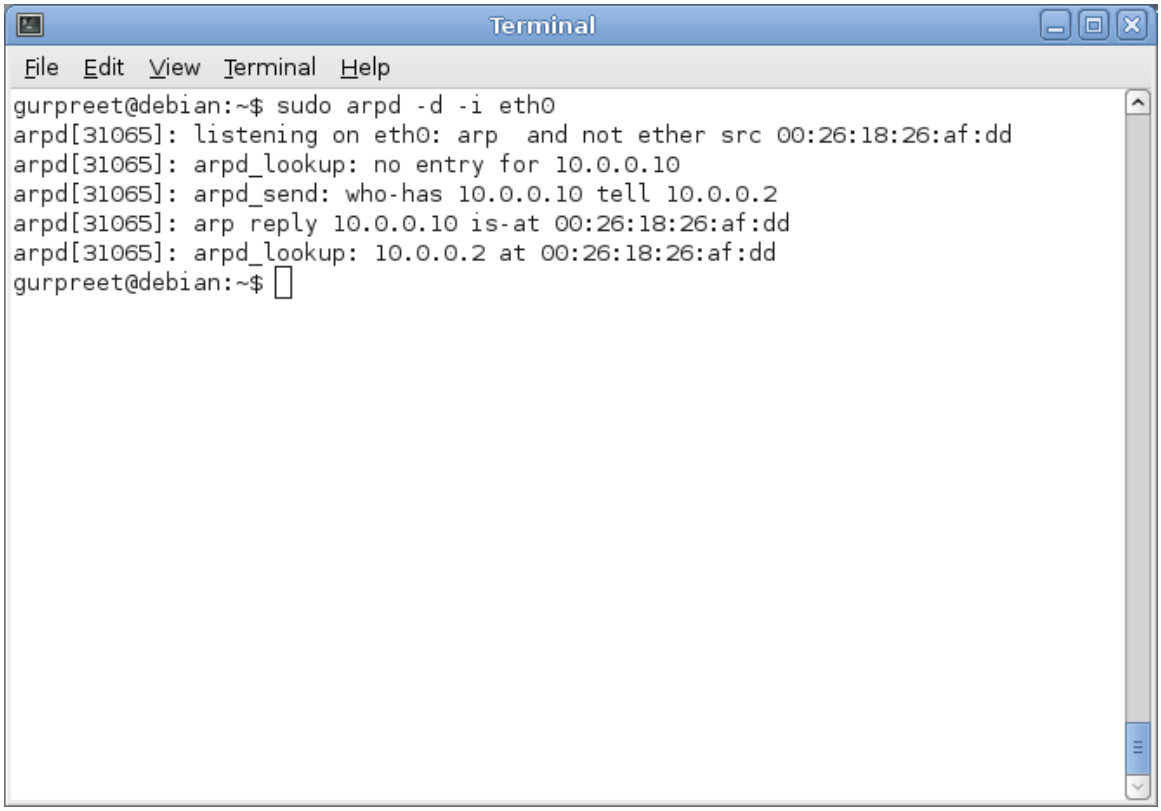

Figure 4.7: arpd responding to ARP request and hence diverting packets to honeyd

Step 3 - Honeyd responding

Honeyd will respond to the ping as shown in Figure 4.8.

Step 4 - Attack gets back ping reply

Attack will get ping reply back from virtual honeypot (Figure 4.9).

4.10.2 Scenario 2 - Attacker telnet-ing virtual honeypot

As port 23 (telnet port) is open on both virtual honeypots, it is an invitation to attackers to connect to the virtual honeypot. The output of Honeyd is shown in Figure 4.10. The output of arpd and ARP request/reply mechanism will remain same and so it is not shown again.

Step 1 - Attacker will run telnet

Attacker will run following command:

telnet 10.0.0.10

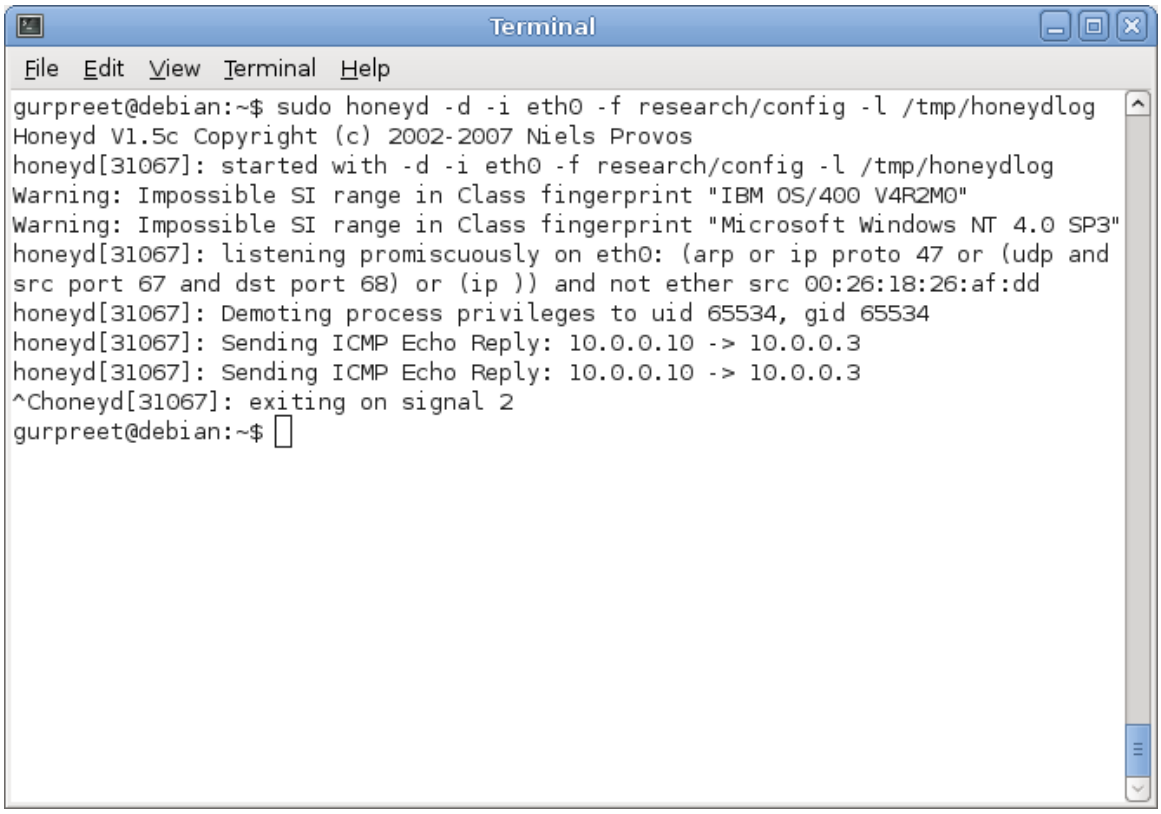

Figure 4.8: Honeyd responding to pings

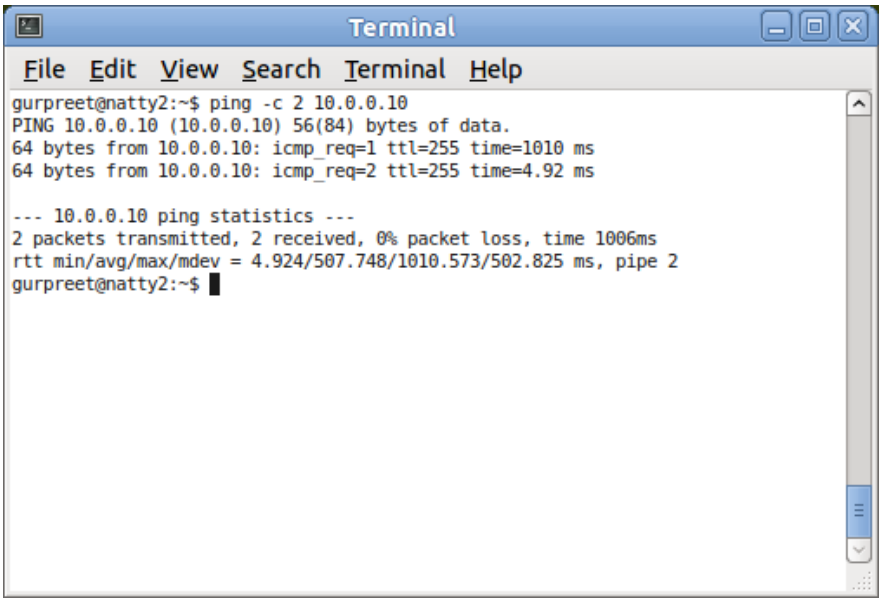

Figure 4.9: Attacker getting back ping reply

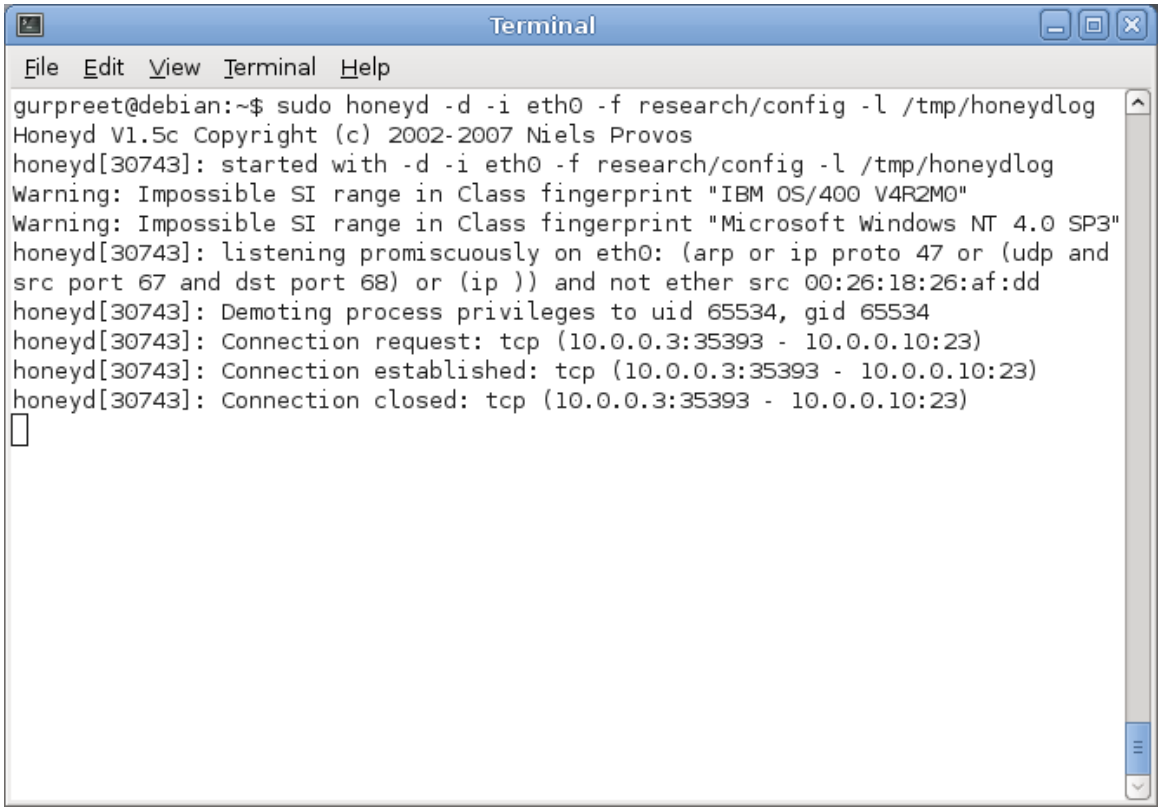

Figure 4.10: Honeyd output when an attacker connects to it and then disconnects

Step 2 - arpd responding

Step 2 will be same (i.e. attacker's system will ask for MAC address and arpd will reply with MAC of system running Honeyd)

Step 3 - Honeyd responding to telnet

Honeyd will respond to telnet and allow attacker to connect to it (Figure 4.10).

Step 4 - Telnet will be connection established

Telnet connection will be established. The attacker's view is shown in Figure 4.11.

4.10.3 Scenario 3 - Attacker trying to find OS of virtual honeypot

Finding remote OS is very crucial for attackers, nmap is very popular for this purpose. Lets see how this honeypot setup stands in that case.

Attacker will run the following command:

sudo nmap -O 10.0.0.11

-O Enables OS detection

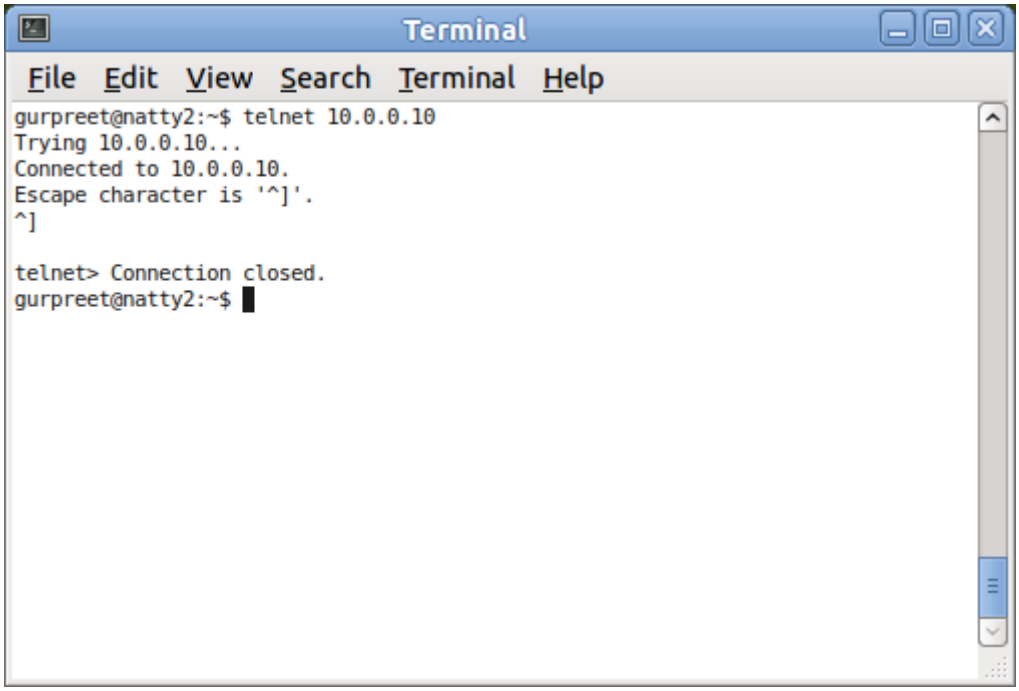

Figure 4.11: Telnet connection established at attacker side

Now nmap will send several probe packets to detect the OS at 10.0.0.10. Arpd will do its usual work and packets will reach Honeyd. Honeyd will make use of Personality Engine to frame the packets as if the reply is coming from TCP stack of Windows and hence nmap will wrongly think that remote machine is running Windows OS when in fact it is running Debian GNU/Linux. Figure 4.12 shows the output of Honeyd after getting all the probes.

Figure 4.13 shows how nmap is fooled when the remote system is running Debian GNU/Linux.

4.11 Problem with using raw honeyd

All the above scenarios used honeyd as a raw program and hence its output is shown on console or is logged to log file. Although it works fine for short runs but once honeyd will be implemented as a campus honeypot and so many curious students start accessing the virtual honeypots the amount of data will become too great to easily see the patterns of the attacks and hence it will defeat the purpose of honeypot.

To cope with this scenario a solution is developed where the output of honeypot is sent to a log file, (this is already being done with -l switch), the log file is read after a fixed interval and is parsed by a python script and this data is uploaded to a remote mysql database.

Once data is in mysql database then it becomes very easy to mine the data. For this purpose a small PHP website is developed which allows the administrator to see the

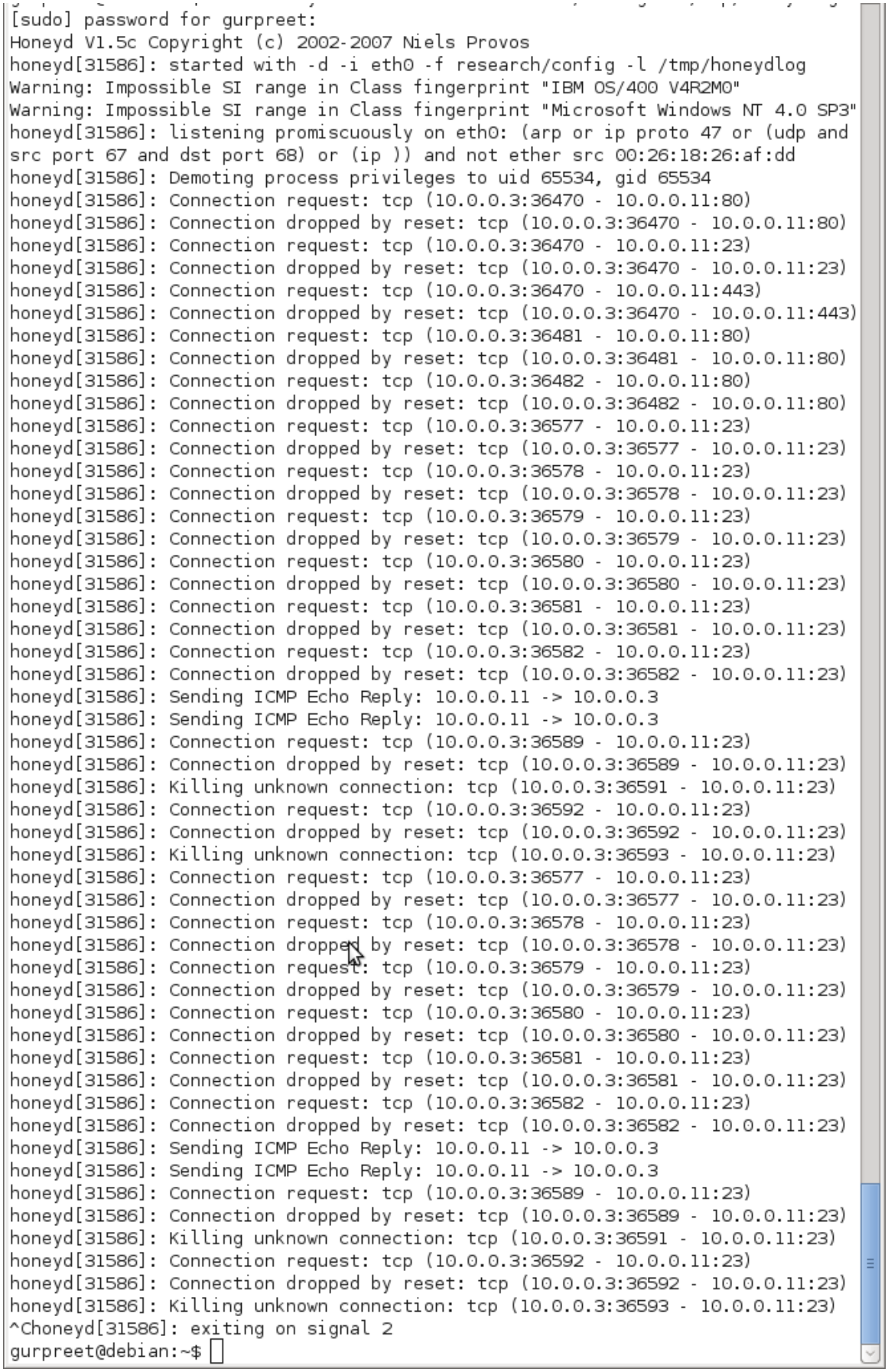

Figure 4.12: Honeyd output when attacker tried to find remote OS

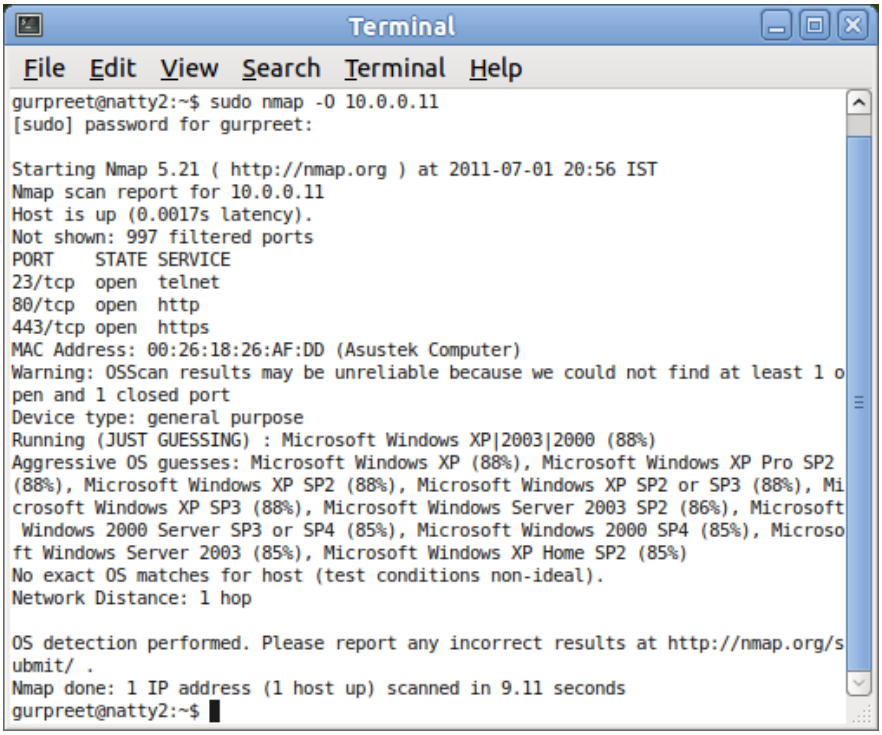

Figure 4.13: Nmap showing remote OS as Windows

Date	Proto	T	Source		Destination		Info	Comment
			IP	Port	IP	Port		
2005-04-02-15:35:15	tcp(6)	S	10.3.6.139	1827	10.1.2.124	3128		[Windows XP SP1]
2005-04-02-15:35:56	tcp(6)	-	10.3.6.139	4378	10.1.2.84	8080:	48 S	[Windows XP SP1]
2005-04-02-15:36:11	tcp(6)	-	10.4.7.196	2671	10.1.2.175	2380:	40 RA	[FreeBSD 5.0-5.1]
2005-04-02-15:39:47	tcp(6)	E	10.3.6.139	1827	10.1.2.123	3128:	9950 240	
2005-04-02-15:40:18	icmp(1)	-	10.3.5.182		10.1.3.99:		11(0): 56	

Figure 4.14: Format of Honeyd's log file
[24]

data and also enables him to search for the required information.

4.12 Parsing the contents of Honeyd log file

In order to parse the log file properly one needs to understand the format of the file. The format of the file is shown in Figure 4.14.

Once the format is understood writing a python script to read the data and insert it into a mysql database is easy. The source code of the python script named 'honeypar.py' is given in Appendix B. The contents of honeyd log file and output of parser script is shown in Figure 4.15. An example run of Honeypar.py script is shown in Figure 4.16.

Each probe is inserted into a mysql table called probes. The format of mysql probes

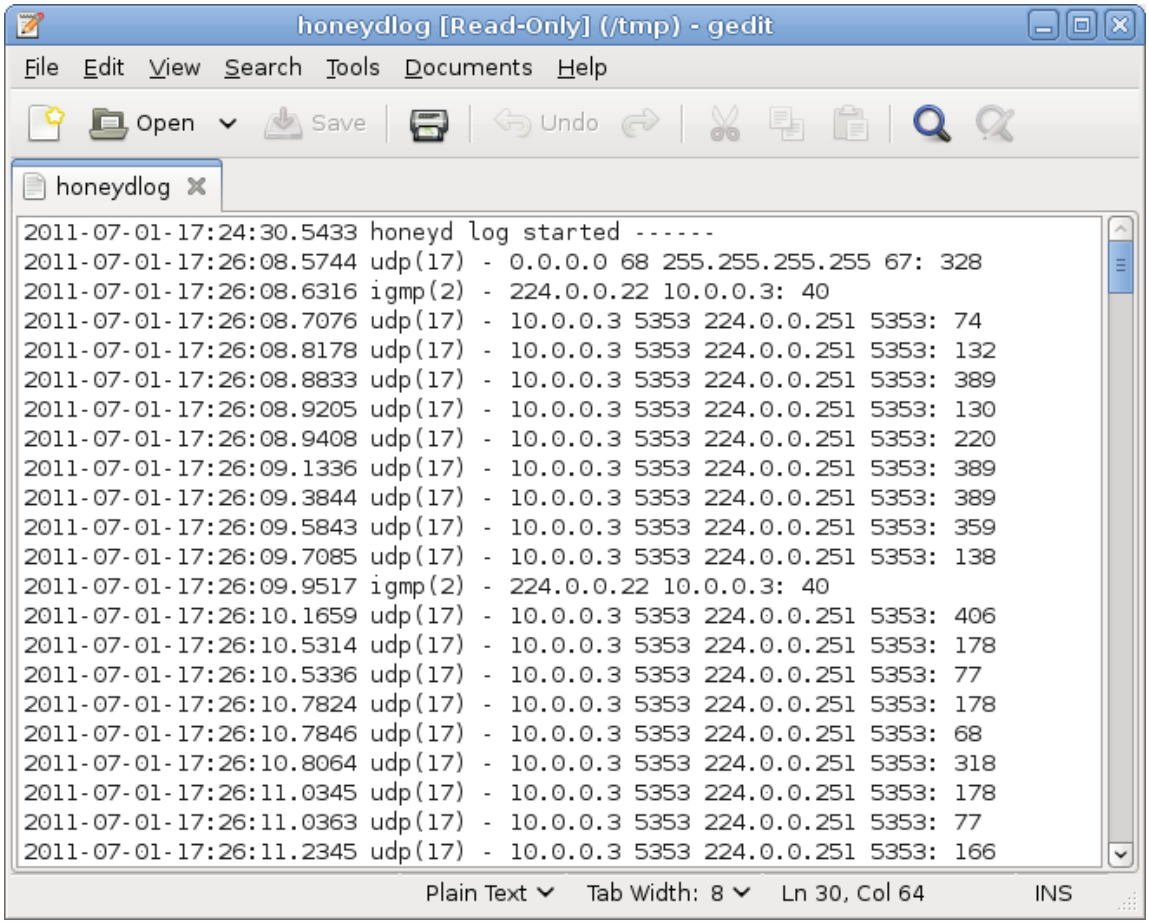

Figure 4.15: Partial contents of Honeyd log file
[24]

table is shown in Figure 4.17.

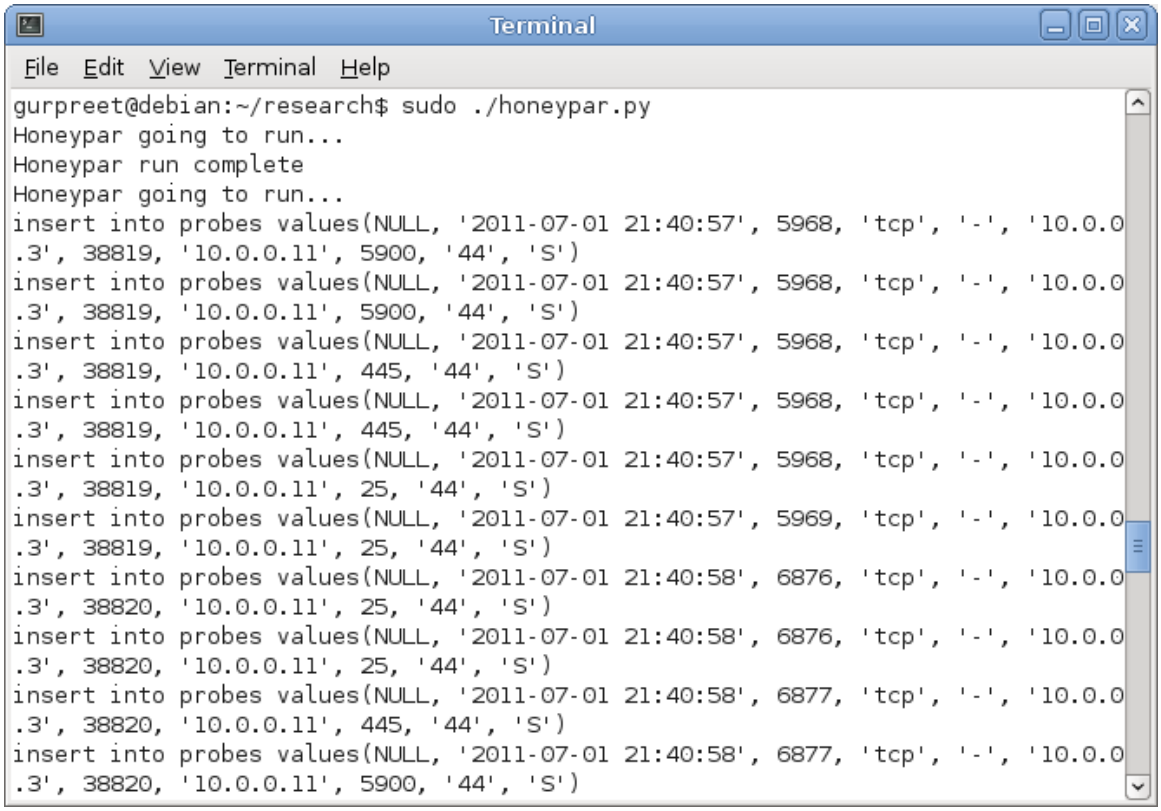

Figure 4.16: Partial output of honeypar.py python script

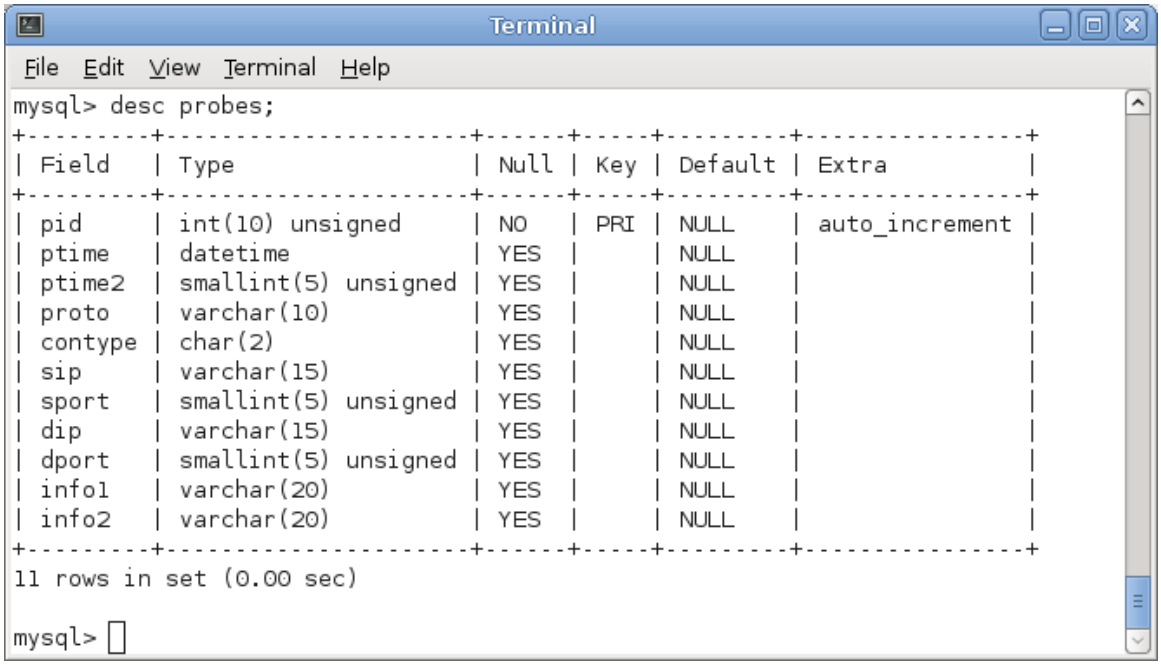

Figure 4.17: MySQL's probes table format

4.13 Displaying the results in a web based interface

It is not easy and intuitive for everyone to query mysql database in order to find some information about the honeypot's runs so to facilitate easy view and searching of data in the mysql database a small and simple website named RemoteHoney is developed in PHP which gets the output from the mysql's probes table and display it in the browser and allows the user to query the database also. Figures 4.18 - 4.22 show various screens of RemoteHoney web based interface.

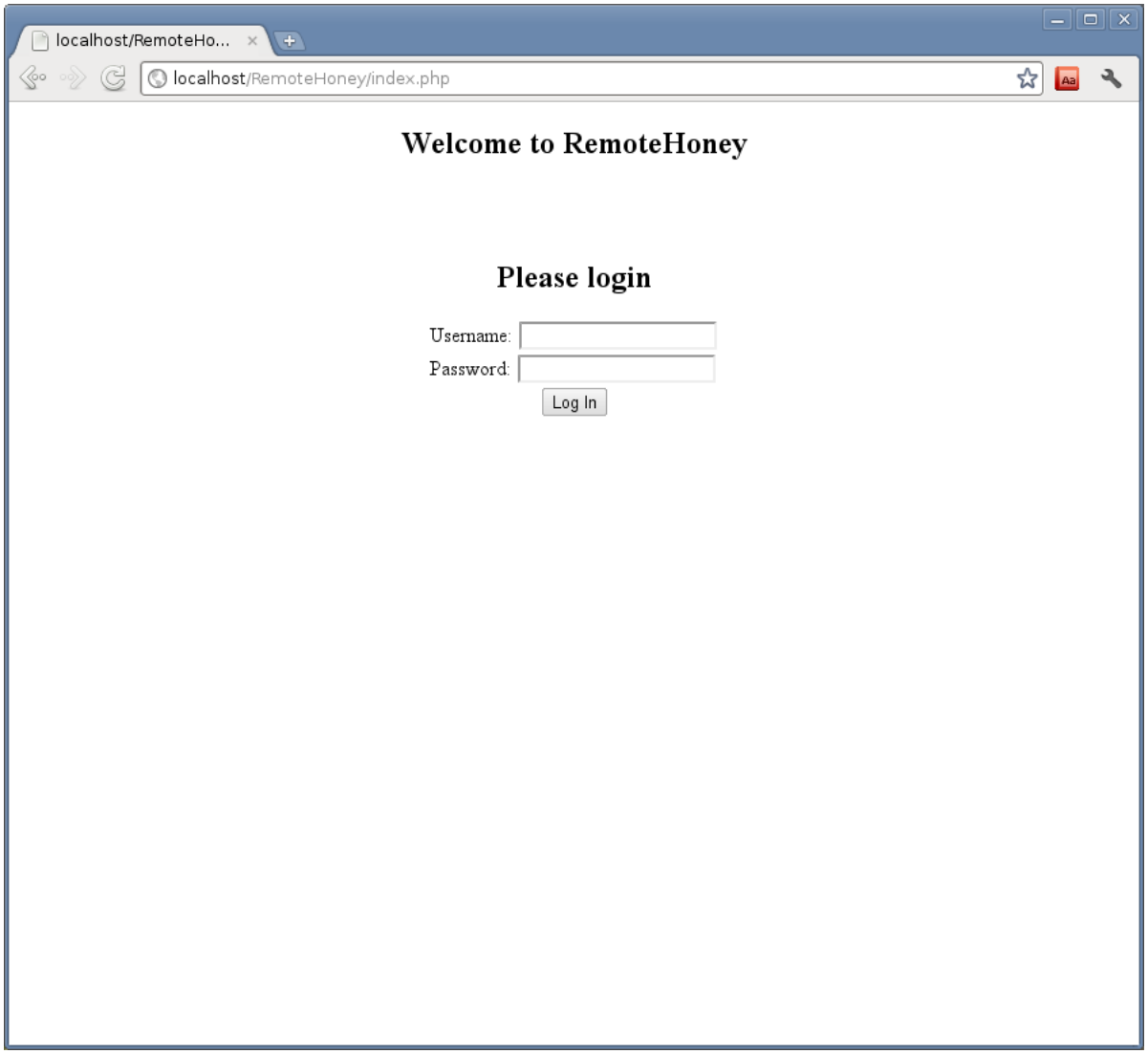

Figure 4.18: The Login Screen of RemoteHoney

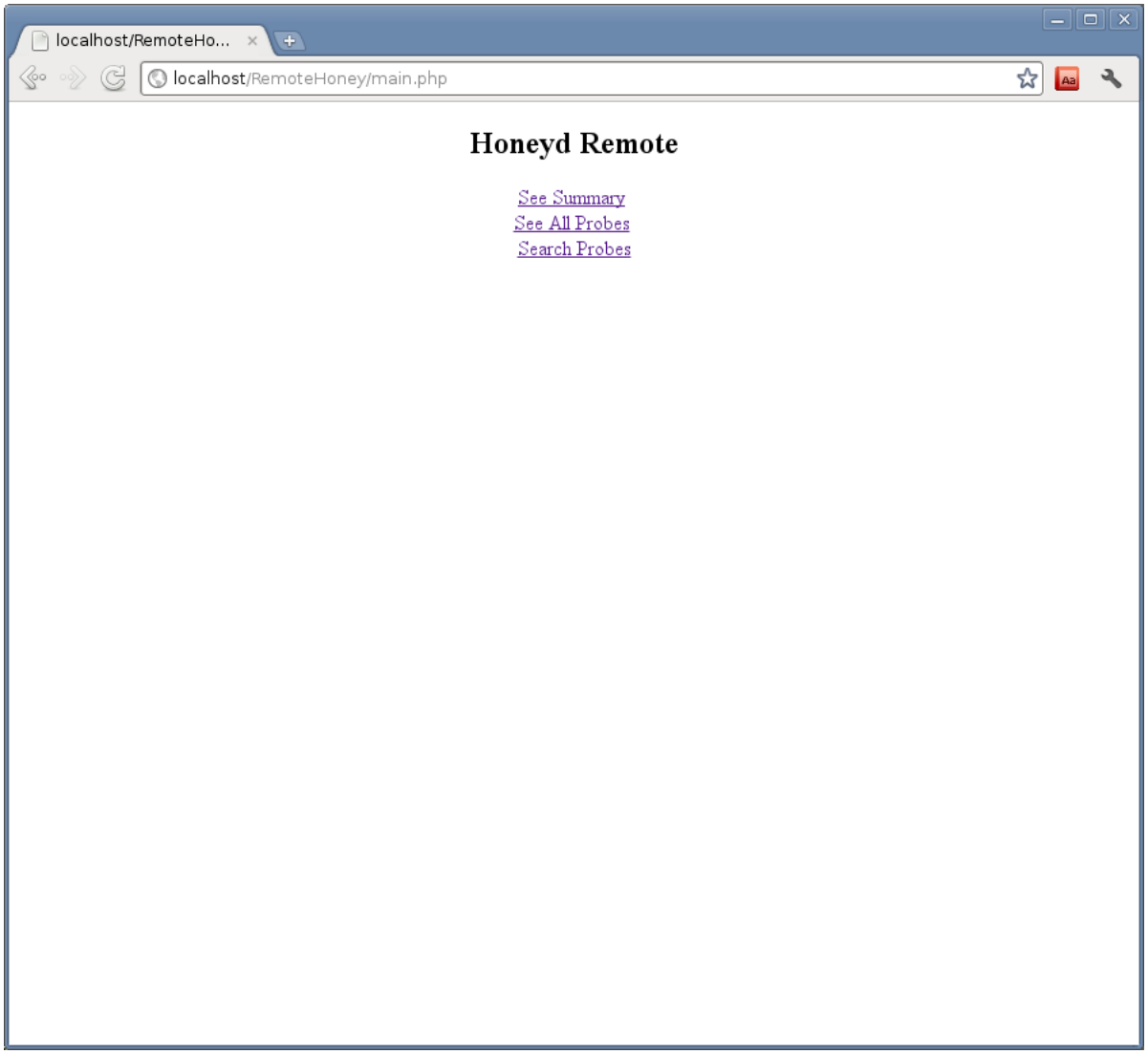

Figure 4.19: Main Page of RemoteHoney

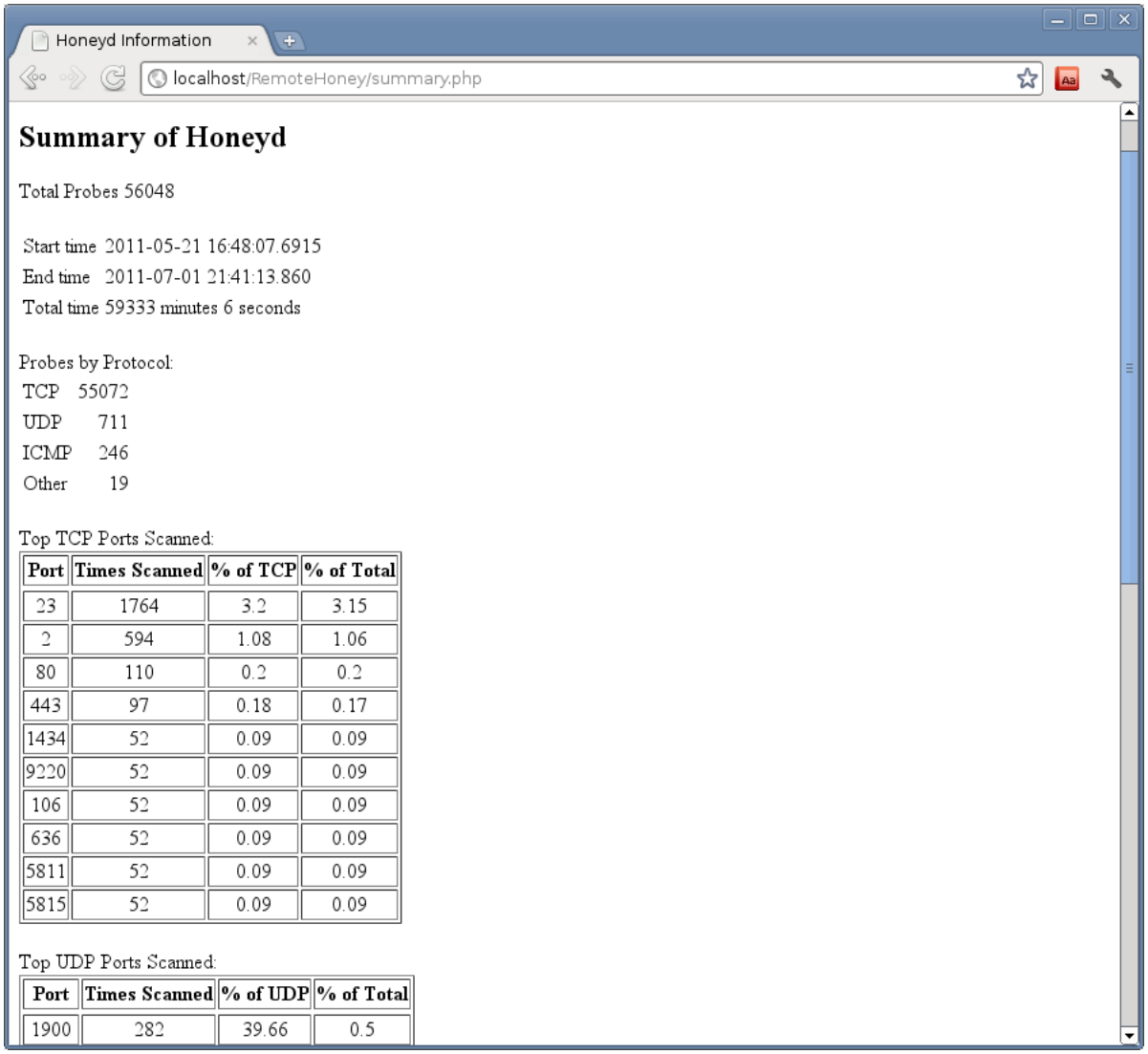

Figure 4.20: Summary page of RemoteHoney

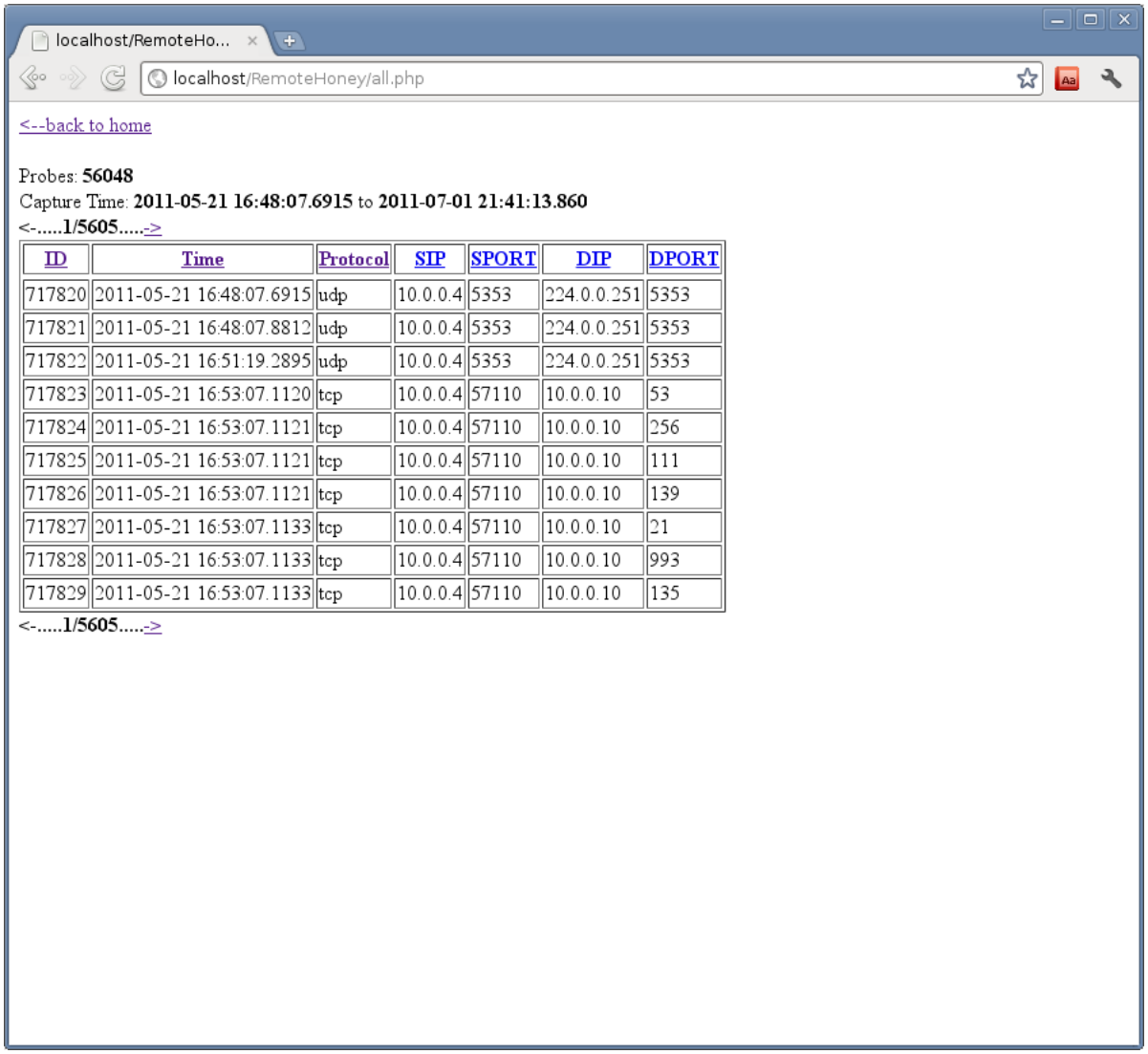

Figure 4.21: Interface to display all probes on Honeypot

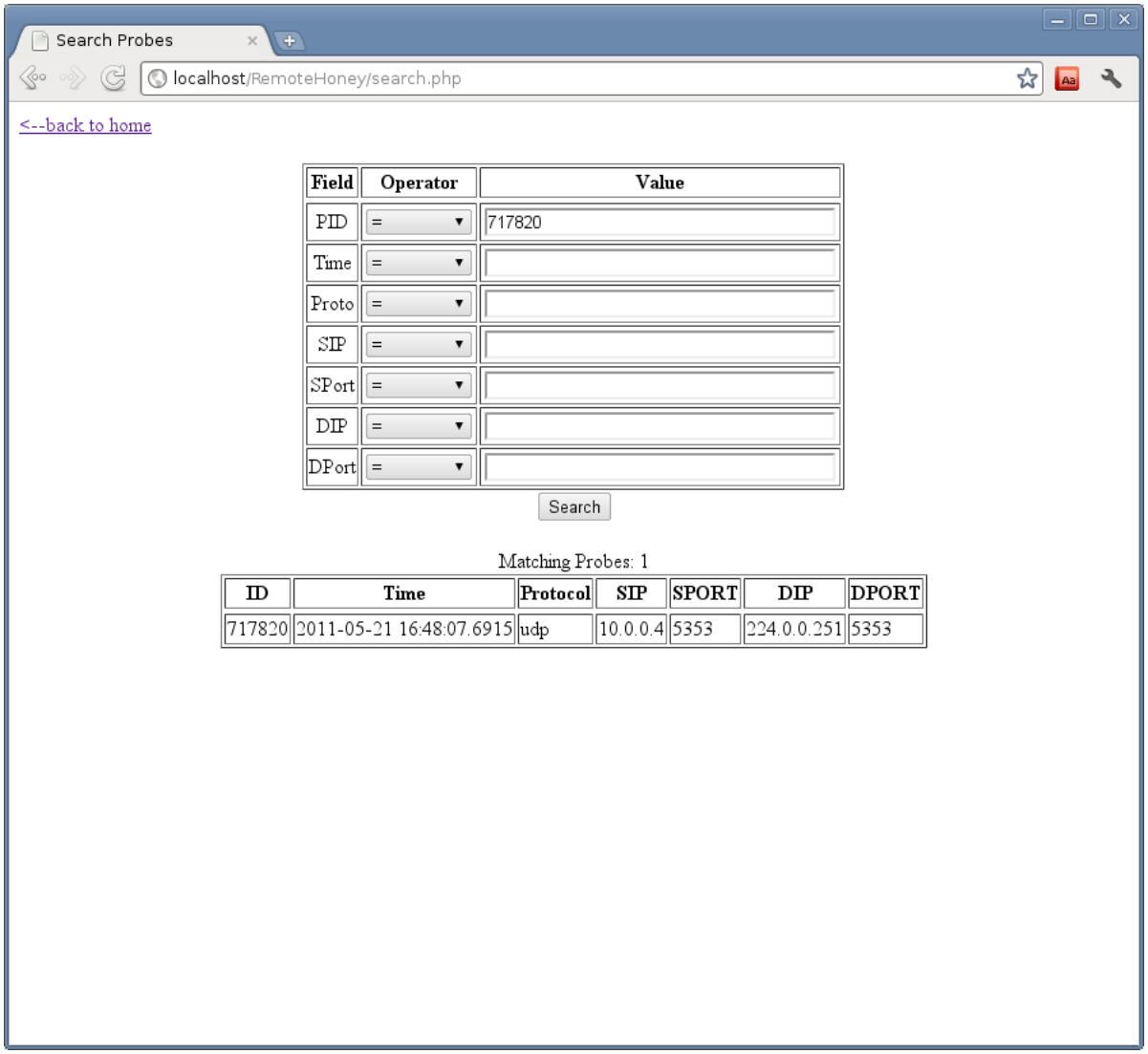

Figure 4.22: Interface to search for probes according to selected criteria

Chapter 5

Results

By combining honeyd, patched arpd, honeypar.py parsing script and RemoteHoney web based interface a complete honeypot solution is ready. Lets perform a test run.

5.1 Test Run

Scenario – Attacker has pinged the remote machine. Campus Honeypot will log everything and administrator can see everything using the web based interface.

Attacker pings the honeypot and gets response

Shown in Figure 5.1.

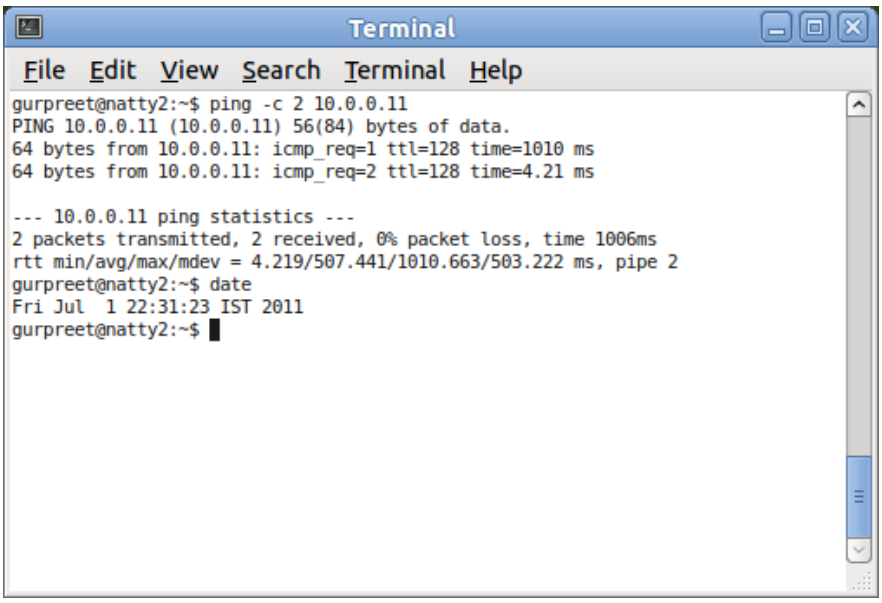

Figure 5.1: Attacker pinging the honeypot and getting response from the honeypot

Arpd does its work

Shown in Figure 5.2.

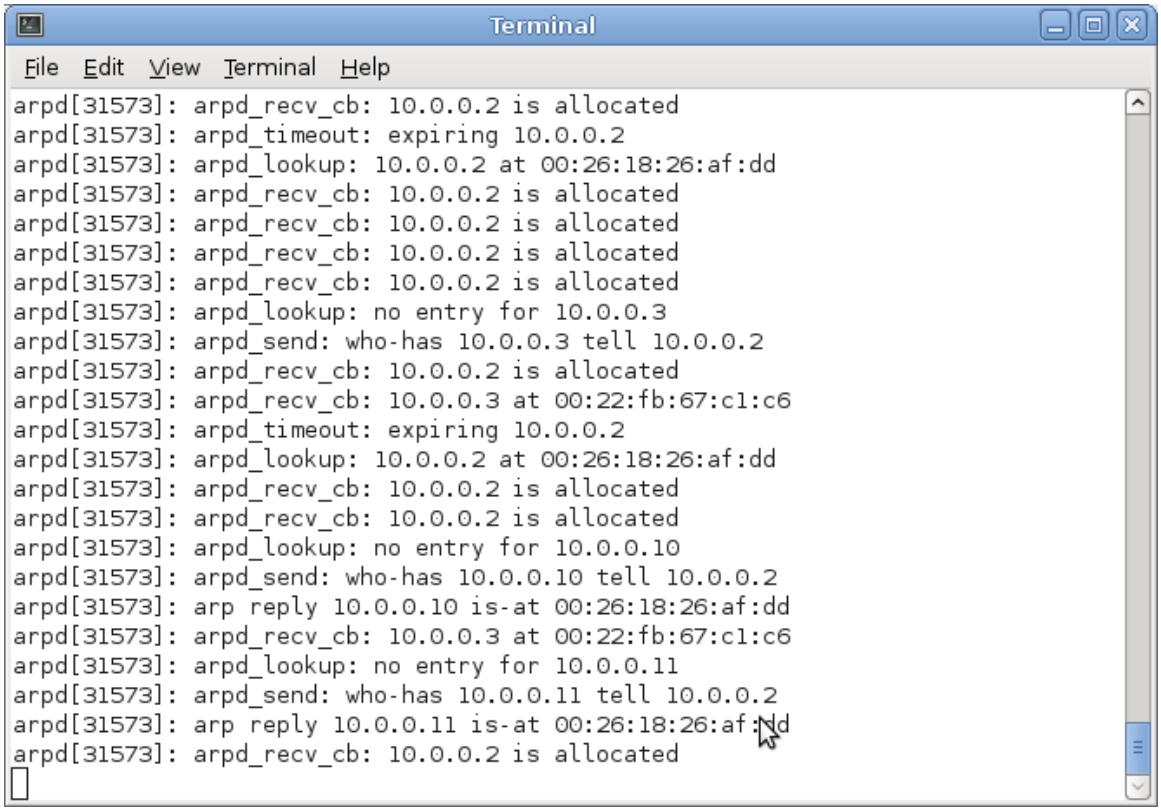

Figure 5.2: Arpd sending back fake ARP replies

Honeyd does its work

Shown in Figure 5.3.

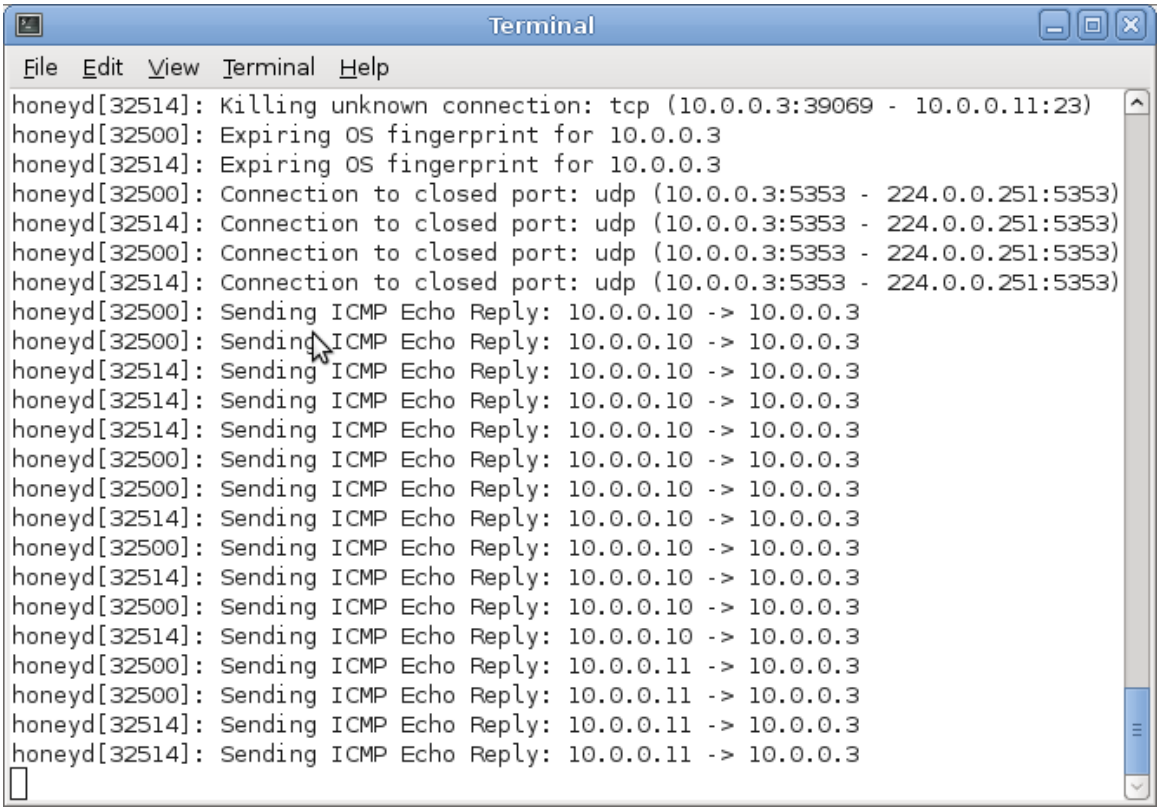

Figure 5.3: Honeyd replying to pings

Honeypar.py does its work

Shown in Figure 5.4.

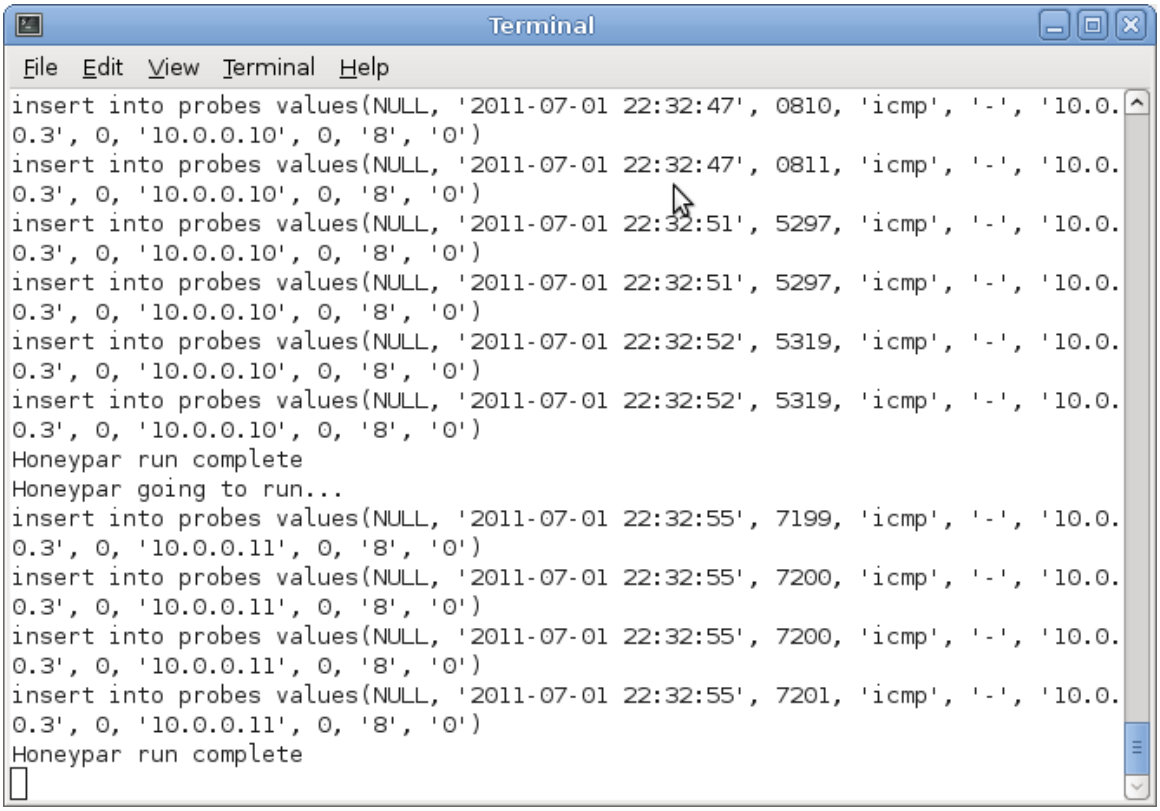

Figure 5.4: Honeypar.py adding probes into database

RemoteHoney's All Interface shows captured probes

Shown in Figure 5.5.

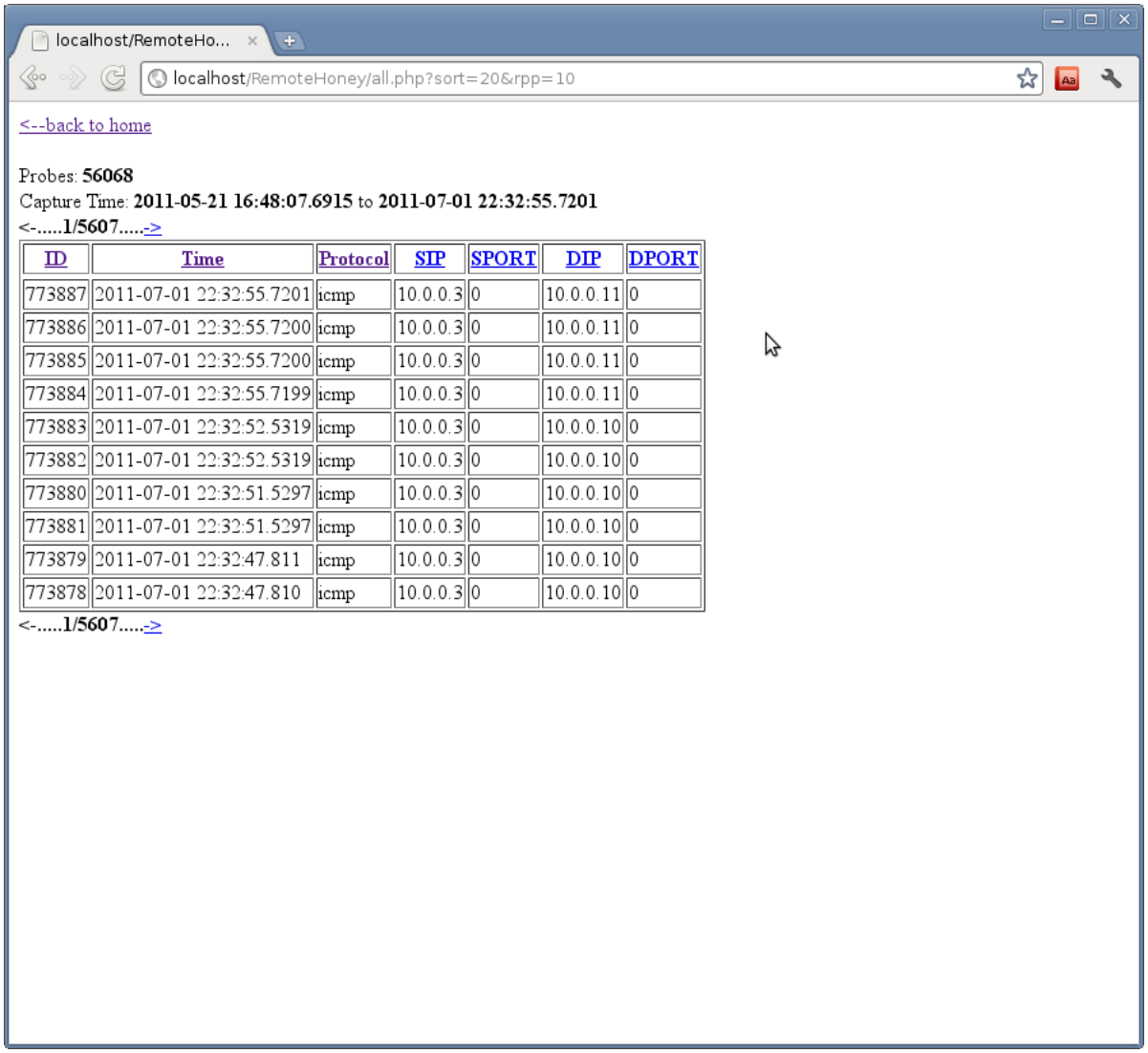

Figure 5.5: Captured probes being shown in RemoteHoney's All Interface (sorted by time desc)

RemoteHoney's Search Interface allows to search for captured probes

Shown in Figure 5.6.

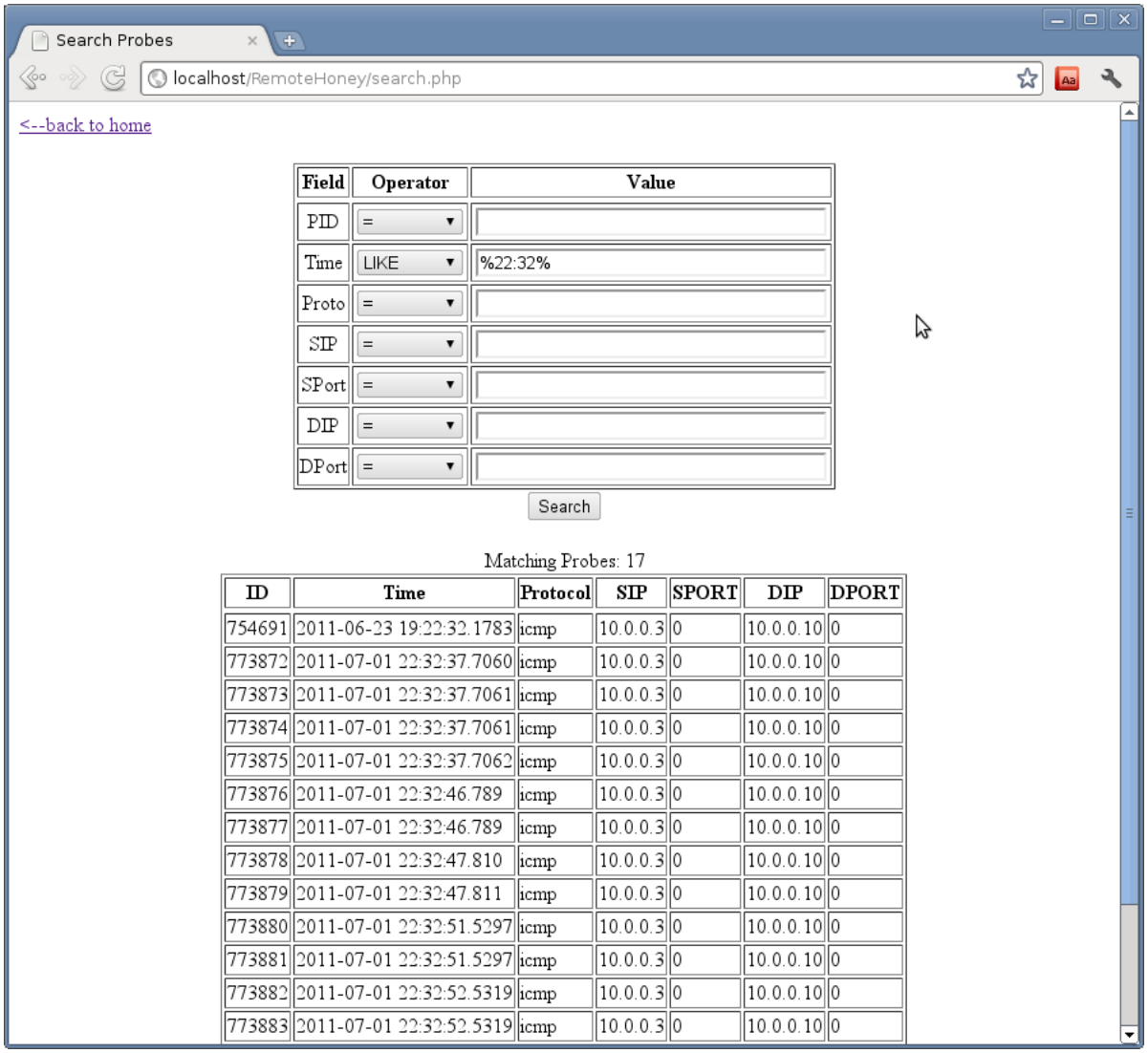

Figure 5.6: Results being shown in RemoteHoney's Search Interface. Searched for time using Like operator of SQL

Chapter 6

Conclusion and Future Scope

6.1 Conclusion

1. A fully functional honeypot solution is developed. Among its main advantages are that it is made up of totally free and open source software.
2. Campus Honeypot enables administrators to install the honeypot once in their campus and then they does not need to do anything except watching the logs remotely using the web based interface.
3. The clever use of broken fingerprint functionality of Honeyd fools nmap's OS probes and hides the true identity of honeypot.
4. The administrator can configure the Honeyd's configuration file according to his/her needs and thus can totally change the behavior of honeypot.
5. Even if honeypot is replaced in place of a popular IP/Server the enormous amount of logs generated should not trouble administrator as he/she can still find out just the feature required, by using the search interface of RemoteHoney web based interface.

6.2 Contributions

1. Developed a Campus Honeypot.
2. Farpd is patched to send unicast ARP replies directed at the original ARP requestor.
3. Farpd also sends out ARP replies faster than before and thus, is able to reply in time i.e. before the process requesting ARP address times out.
4. Found a work around, to deal with broken fingerprint functionality of Honeyd.

6.3 Future Scope

1. Although Honeyd's fingerprints are used in this work but in affect they are broken because the fingerprints of nmap no longer matches that of honeyd. Hence, honeyd's personality engine needs to be rewritten encorporating new/changed scans and their responses.

2. Farpd is patched to send unicast replies and to send ARP replies faster by waiting for no answer to ARP request, only once. Although the farpd seems to be working fine in this work, the full ramifications of these changes needs to be properly understood which demands a full study on its own.

References

[1] Online Etymology Dictionary, http://www.etymonline.com/index.php?term=secure. Fetched 15/06/2011

[2] The Good Security Recipe, http://blog.rootshell.be/2010/08/09/the-good-security-recipe/. Fetched 15/06/2011

[3] Hacker, http://en.wikipedia.org/wiki/Hacker_(computer_security). Fetched 15/06/2011

[4] The Five Phase Approach of Malicious Hackers, http://blog.phpkemist.com/2008/07/20/the-five-phase-approach-of-malicious-hackers/. Fetched 15/06/2011

[5] Ethical Hacking, Slide 7, cehlab.com/excisefiles/CEH-presentation/Module%201%20V%203.0.ppt. Fetched 15/06/2011

[6] Barry M. Leiner, Vinton G. Cerf, Robert E. Kahn, John Postel et al., *A Brief History of the Internet*. ACM SIGCOMM Computer Communication Review, Volume 39, Number 5, October 2009.

[7] Internet map, http://en.wikipedia.org/wiki/File:Internet_map_1024.jpg. Fetched 24/06/2011

[8] Behrouz A. Forouzan, *Data Communications and Networking*. Tata McGraw-Hill, 2004

[9] OSI 7 Layers Pic, http://www.novell.com/info/primer/art/prim02.gif. Fetched 24/06/2011

[10] OSI 7 Layers, http://en.wikipedia.org/wiki/OSI_model. Fetched 18/06/2011

[11] TCP/IP model, http://en.wikipedia.org/wiki/TCP/IP_model. Fetched 18/06/2011

[12] TCP/IP model, http://www.tcpipguide.com/free/diagrams/tcpiplayers.png. Fetched 19/06/2011

[13] RFC 791 (Internet Protocol), http://tools.ietf.org/html/rfc791. Fetched 19/06/2011

[14] TCP/IP Reference, http://nmap.org/book/tcpip-ref.html. Fetched 19/06/2011

[15] Transmission Control Protocol, http://en.wikipedia.org/wiki/Transmission_Control_Protocol. Fetched 19/06/2011

[16] RFC 768 (User Datagram Protocol), http://tools.ietf.org/html/rfc768. Fetched 20/06/2011

[17] User Datagram Protocol, http://en.wikipedia.org/wiki/User_Datagram_Protocol. Fetched 20/06/2011

[18] RFC 792 (Internet Control Message Protoco), http://tools.ietf.org/html/rfc792. Fetched 20/06/2011

[19] Internet Control Message Protoco, http://en.wikipedia.org/wiki/Internet_Control_Message_Protocol. Fetched 20/06/2011

[20] Lance Spitzner, *Honeypots: Catching the Insider Threat*. Computer Security Applications Conference, 19th Annual, 2003

[21] Eric Peter et al., *A Practical Guide to Honeypots*. http://www.cse.wustl.edu/~jain/cse571-09/ftp/honey/index.html, Fetched 20/06/2011.

[22] Neils Provos, *A Virtual Honeypot Framework*. SSYM'04 Proceedings of the 13th conference on USENIX Security Symposium, Volume 13, 2004

[23] Lance Spitzner, *Honeypots: Tracking Hackers*. Addison Wesley, September 13, 2002

[24] Neils Provos,Thorsten Holz, *Virtual Honeypots: From Botnet Tracking to Intrusion Detection*. Addison Wesley Professional, July 16, 2007

[25] Ryan Talabis, *Honeypots 101: What's in it for me?*. http://www.philippinehoneynet.org/index.php?option=com_docman&task=doc_download&gid=3&Itemid=29, 2007, Fetched 21/06/2011

[26] Ryan Talabis, *Honeypots 101: Risks and Disadvantages*. http://www.philippinehoneynet.org/index.php?option=com_docman&task=doc_download&gid=4&Itemid=29. 2007, Fetched 21/06/2011

[27] Ryan Talabis, *Honeypots 101: A Brief History of Honeypots*. http://www.philippinehoneynet.org/index.php?option=com_docman&task=doc_download&gid=2&Itemid=29. 2007, Fetched 21/06/2011

[28] Cliff Stoll, *The Cuckoo's Egg*. Doubleday Publishing Group, 1989

[29] Bill Cheswick, *An Evening with Berferd in which a cracker is Lured, Endured, and Studied*. In Proc. Winter USENIX Conference, 1992

[30] Deception Toolkit UI, http://all.net/dtk/gui.jpg. Fetched 21/06/2011

[31] HoneyBOT User Guide, www.atomicsoftwaresolutions.com/HoneyBOTUserGuide.pdf. Fetched 24/06/2011

[32] Specter: A Commercial Honeypot Solution for Windows, http://www.symantec.com/connect/articles/specter-commercial-honeypot-solution-windows. Fetched 22/06/2011

[33] Deception Toolkit, http://all.net/dtk/index.html. Fetched 23/06/2011

[34] CyberCop Sting Getting Started Guide, Version 1.0, Networks Associates Technology, Inc May 1999

[35] NetFacade Intrusion Detection, http://www22.verizon.com/fns/solutions/netsec/netsec_netfacade.html. Fetched 23/06/2011

[36] Know Your Enemy: Sebek, http://www.securitydocs.com/library/2769/. Fetched 23/06/2011

[37] La Brea README, http://labrea.sourceforge.net/README. Fetched 24/06/2011

[38] Honeytokens: The Other Honeypot, http://www.symantec.com/connect/articles/honeytokens-other-honeypot. Fetched 24/06/2011

[39] Know Your Enemy:Honeynets, http://old.honeynet.org/papers/honeynet/. Fetched 24/06/2011

Publications

Gurpreet Singh, Dr. Maninder Singh "*Design and Development of a Honeypot Solution for Campus Network*", *International Journal of Engineering Science and Technology* ISSN: 0975-5462 (Communicated)

Appendix A

autoconfig.py Script

Following Python script is used to automatically generate a large configuration file containing all the personalities possible for Honeyd. Because personality engine of Honeyd is output, this script was used to find which of the personalities of Honeyd are recognized by Nmap as Linux and which are recognized as Windows.

This script uses the help of two additional files 'ips.txt' and 'pers.txt'. The former file contain a list of IP addresses which will be assigned to personlities (Figure A.1). The latter file contain all the personalities grepped from honeyd's os fingerprint file which by default resides at location '*/usr/local/share/honeyd/nmap.prints*' (Figure A.2).

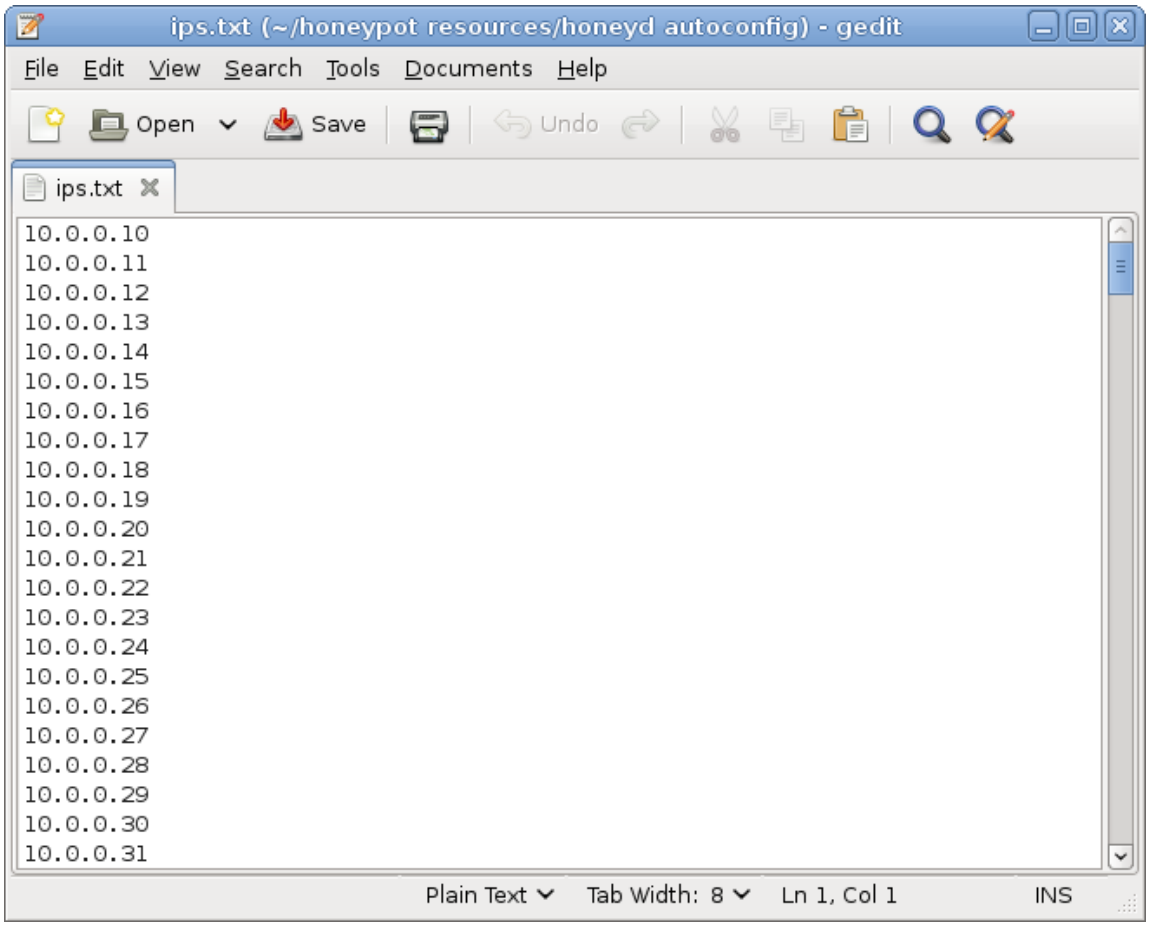

Figure A.1: IP Address for use by the Autoconfig.py Script

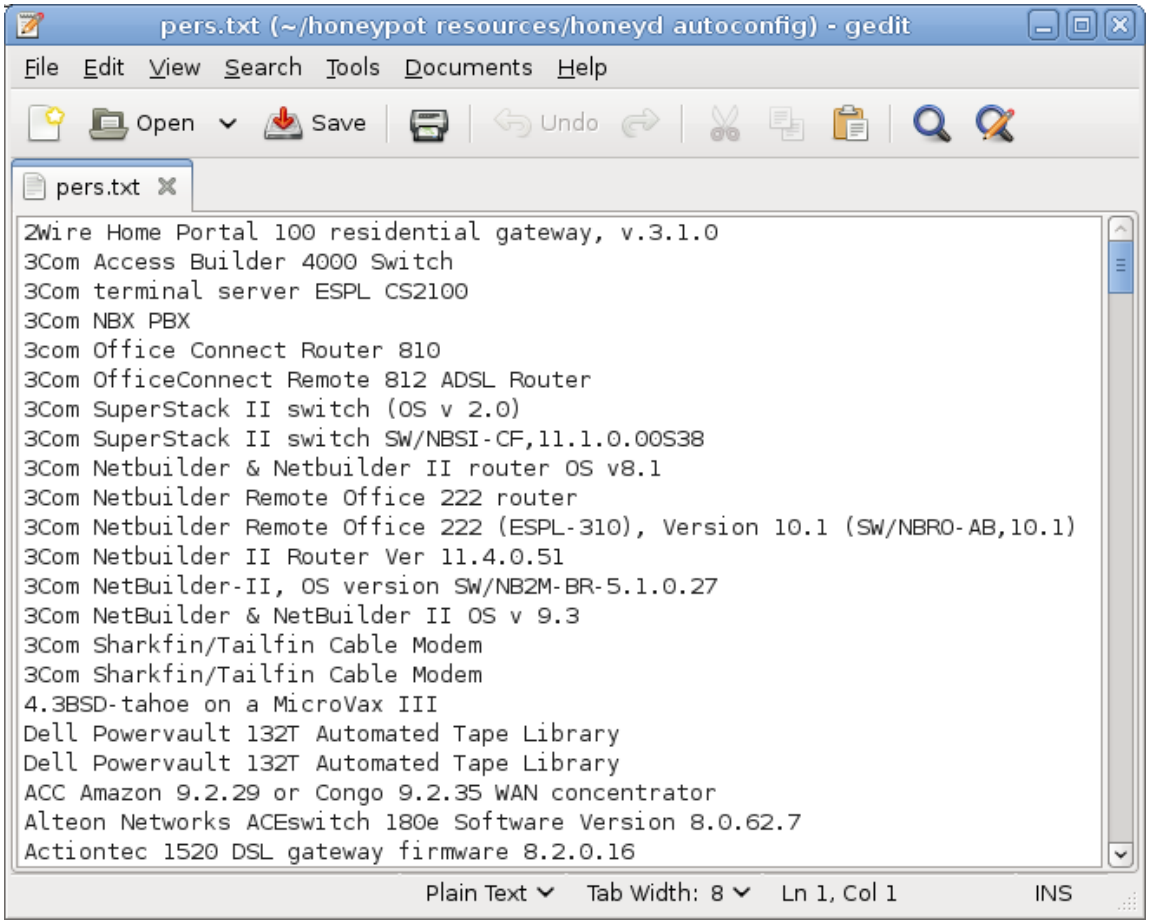

Figure A.2: Personalities for use by the Autoconfig.py Script

The script's source code is as follows:

```
#!/usr/bin/python

def Start():
        conffile = open('autoconfig','w')
        persfile = open('pers.txt','r')
        ipsfile  = open('ips.txt','r')

        conffile.write("create template\n")
        conffile.write("set template default tcp action reset\n")
        conffile.write("set template default udp action reset\n")
        conffile.write("set template default icmp action open\n")
        conffile.write("add template tcp port 23 open\n")
        conffile.write("add template tcp port 443 open\n")
        conffile.write("add template tcp port 80 reset\n\n")
        for line in persfile:
                pers = line[0:-1]
```

```
                line1 = 'set template personality "' + pers + '"\n'
                conffile.write(line1)

                ip = ipsfile.readline()
                ip = ip[0:-1]
                line2 = 'bind ' + ip + ' template\n\n'
                conffile.write(line2)

if __name__ == '__main__':
        Start();
```

The output file named 'autoconfig' is generated by the script. Figure A.3 shows partial contents of the file.

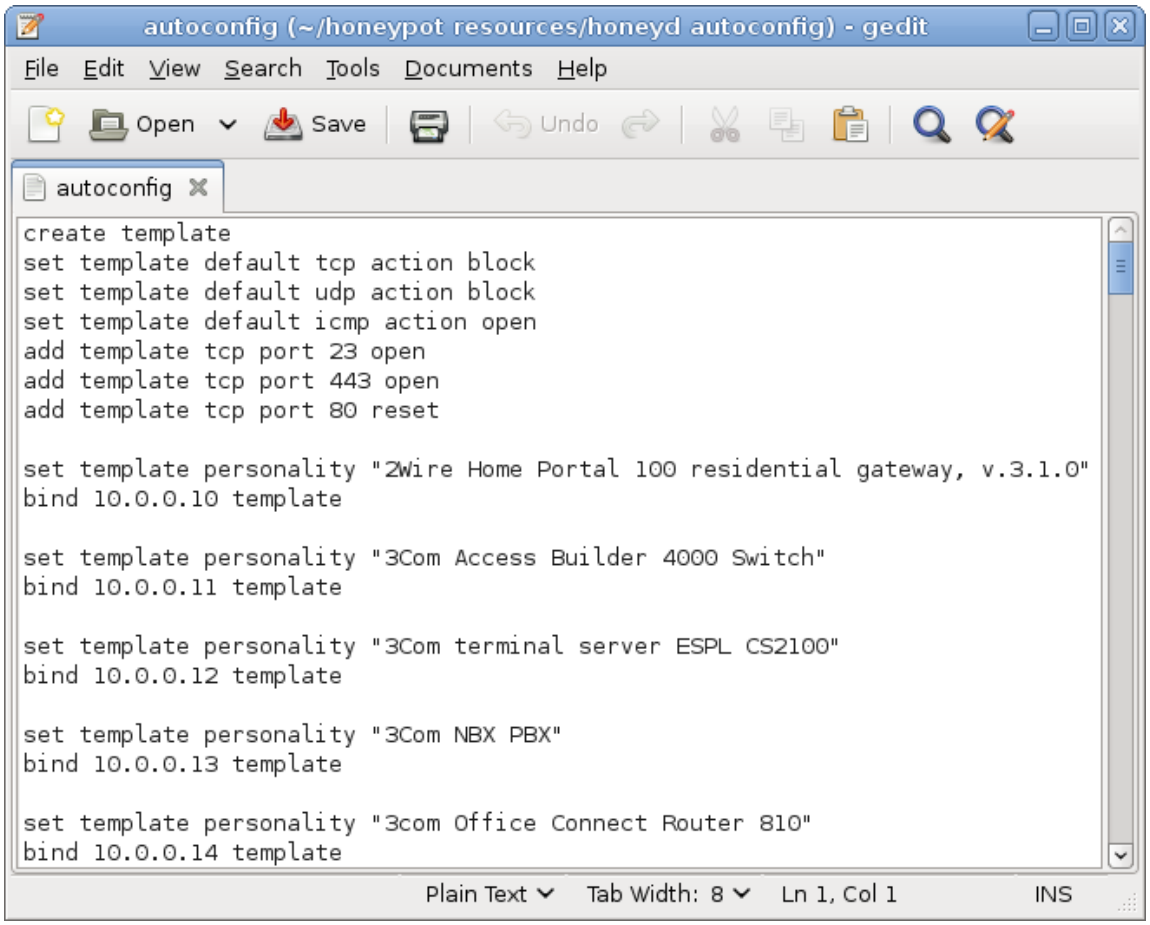

Figure A.3: Output of the Autoconfig.py Script

Appendix B

honeypar.py Script

Following Python script is used to parse Honeyd log file and insert its contents in the mysql database. The source code of the script is given below:

```
#!/usr/bin/env python
import MySQLdb
import time
import os
def Log(t,cur):
    sql = "insert into probes values(NULL, " + \
     "'"+t[0]+"', " + \
     str(t[1])+", "      + \
     "'"+t[2]+"', " + \
     "'"+t[3]+"', " + \
     "'"+t[4]+"', " + \
     str(t[5])+", "      + \
     "'"+t[6]+"', " + \
     str(t[7])+", "      + \
     "'"+t[8]+"', " + \
     "'"+t[9]+"')"
    print sql
    cur.execute(sql)

def parse_tcp(line,cur):
     fields = line.split(" ")

     time = fields[0]
     date = time.split(".")[0]
     hxus = time.split(".")[1]
     #change "YYYY-MM-DD-HH:MM:SS" to "YYYY-MM-DD HH:MM:SS"
     date = date[:date.rfind("-")] + " " + date[date.rfind("-")+1:]
```

```
    proto = fields[1].split("(")[0]
    ctype = fields[2]
    sip = fields[3]
    sport = int(fields[4])
    dip = fields[5]
    dport = fields[6]
    info1 = "-"    #size
    info2 = "-"    #flags
    if len(fields) >= 9:
        dport = int(dport[:-1])
        if fields[7] != "":
            info1 = fields[7]
        if fields[8] != "":
            info2 = fields[8]
    Log((date,hxus,proto,ctype,sip,sport,dip,dport,info1,info2),cur)

def parse_udp(line,cur):
    fields = line.split(" ")

    time = fields[0]
    date = time.split(".")[0]
    hxus = time.split(".")[1]
    #change "YYYY-MM-DD-HH:MM:SS" to "YYYY-MM-DD HH:MM:SS"
    date = date[:date.rfind("-")] + " " + date[date.rfind("-")+1:]

    proto = fields[1].split("(")[0]
    ctype = fields[2]
    sip = fields[3]
    sport = int(fields[4])
    dip = fields[5]
    dport = fields[6]
    info1 = "-"  #size
    info2 = "-"  #don't exist
    if len(fields) >= 8:
        dport = int(dport[:-1])
        if fields[7] != "":
            info1 = fields[7]
    Log((date,hxus,proto,ctype,sip,sport,dip,dport,info1,info2),cur)

def parse_icmp(line,cur):
```

```
    fields = line.split(" ")

    time = fields[0]
    date = time.split(".")[0]
    hxus = time.split(".")[1]
    #change "YYYY-MM-DD-HH:MM:SS" to "YYYY-MM-DD HH:MM:SS"
    date = date[:date.rfind("-")] + " " + date[date.rfind("-")+1:]

    proto = fields[1].split("(")[0]
    ctype = fields[2]
    sip = fields[3]
    sport = "0"
    dip = fields[4]
    dport = "0"
    if dip[-1:]== ":":
        dip = dip[:-1]
    info1 = fields[5].split("(")[0]  #icmp code
    info2 = fields[5].split("(")[1][:-2]  #icmp type
    Log((date,hxus,proto,ctype,sip,sport,dip,dport,info1,info2),cur)

def parse_igmp(line,cur):
    fields = line.split(" ")

    time = fields[0]
    date = time.split(".")[0]
    hxus = time.split(".")[1]
    #change "YYYY-MM-DD-HH:MM:SS" to "YYYY-MM-DD HH:MM:SS"
    date = date[:date.rfind("-")] + " " + date[date.rfind("-")+1:]

    proto = fields[1].split("(")[0]
    ctype = fields[2]
    sip = fields[3]
    sport = "0"
    dip = fields[4]
    dport = "0"
    if dip[-1:]== ":":
        dip = dip[:-1]
    info1 = fields[5] #igmp size
    info2 = "-"
    Log((date,hxus,proto,ctype,sip,sport,dip,dport,info1,info2),cur)
```

```
def parse_other(line,cur):
    raise RuntimeError("Unhandled Protocol Found - "+ line)

if __name__ == "__main__":
    fname = "/tmp/honeydlog"
    cxn = MySQLdb.connect(host='127.0.0.1', db='honeyd', \
        user='honeyu', passwd='honeyp')
    cur = cxn.cursor()
    while True:
        print "Honeypar going to run..."
        os.system("killall -STOP honeyd")
        for line in open(fname):
            if line.find("honeyd log") == -1:
                line=line[:-1]
                proto = line.split(" ")[1]
                if proto.find('tcp') != -1:
                    parse_tcp(line,cur)
                elif proto.find('udp') != -1:
                    parse_udp(line,cur)
                elif proto.find('icmp') != -1:
                    parse_icmp(line,cur)
                elif proto.find('igmp') != -1:
                    parse_igmp(line,cur)
                else:
                    parse_other(line,cur)
        f = open(fname,'w')
        f.truncate(0)
        os.system("killall -CONT honeyd")
        print "Honeypar run complete"
        time.sleep(60)
    cur.close()
    cxn.close()
```