An Analysis of Network Attacks and their Countermeasures

Ahmed Obied
Department of Computer Science
University of Calgary

April 15, 2005

Abstract

Malicious attacks are getting smarter, more widespread and increasingly difficult to detect, and dozens more are added to the menagerie each day. Identifying and classifying the type of malicious program spreading across global networks is a crucial step in developing strategies to contain and eradicate it. This paper describes the most common attacks used these days to paralyze computer and network resources. It provides network traces of malicious traffic and strategies for providing better countermeasures. Data sets captured at the University of Calgary are analzyed to classify intrusion attempts and identify security holes in the University's network.

Keywords: Computer Security, Network Security, Internet Security, Cryptography, Networking, Intrusion Detection, Firewalls, Attackers, Network Attacks

1 Introduction

Computers are a must in today's highly technical world. They are used in schools, universities, hospitals, military, space shuttles and many more. Their usage increased tremendously in the last few years and millions of users joined this technological revolution due to the use of the Internet that made the world look so small and at our disposal. However, the widespread use of the Internet caused the number of warnings being made about the dark side of our technological revolution to increase and we are becoming uniquely vulnerable to many mysterious and malicious threats. Malicious software has been and still used to spread mayhem, enact political revenge on a corporate target, steal data, increase access to a network resource, hijack networks, deny companies use of their network, or sometimes simply gain bragging rights.

Devising effective defenses against network attacks is a difficult and elusive goal for system administrators and information security researchers. The wide range of computer hardware, the complexity of operating systems, the variety of potential vulnerabilities, and the skill of many attackers combine to create a problem that is extremely difficult to address [2]. To be able to accurately identify, classify, and provide better countermeasures against the many types of malicious threats, one must first understand how the authors of malware think, what motivates them, and what are the tools and techniques they use. This paper describes the background literature needed to understand many network attacks, discusses the types of people involved in writing malicious code, and gives a detailed description of the common attacks used these days along with some traces and analysis of real data sets.

2 Background

Essential background literature to understand the foundations of many network attacks and their countermeasures are described below:

2.1 Networking

Is it possible to implement a security solution without understanding TCP/IP? The answer to this question is no. TCP/IP stands for the Transmission Control Protocol/Internet Protocol. It is a suite for a number of protocols that are required to allow hosts on the Internet to communicate. The following four layers comprise the TCP/IP Internet model [9]:

1. **Application layer:** Handles implementation of user applications.
2. **Transport layer:** Manages end-to-end communications between hosts.
3. **Network layer:** Gets data from source to destination.
4. **Link layer:** Manages data transfer to and from physical medium.

The most relevant and important network protocols are described below:

- **IP:** The Internet Protocol is used at the Network layer. IP is responsible for getting a packet from the source IP address to the destination IP address where

such addresses are defined in the IP header. Important fields in the IP header are: the protocol number field which identifies what protocol is used (TCP, UDP, or ICMP), the flags and fragment offset fields that are used to keep track of the fragments when a datagram has to be split up into smaller packets, and the time to live (TTL) field which is a number that is decremented whenever the datagram passes through a router.

- **TCP:** The Transmission Control Protocol is a connection oriented protocol that offers the application (client) a reliable connection to a program (server) on another machine. Once this connection is established, the TCP protocol guarantees the correct (both in content and in order) delivery of the data transmitted through this circuit.

  A TCP header follows the IP header (if the TCP protocol is to be used) supplying information specific to the TCP protocol. TCP is a sliding window protocol with time-outs and retransmits. Outgoing data must be acknowledged by the far-end TCP. Acknowledgments can be piggybacked on data. Important fields in the TCP header are: the source and destination port numbers which identify services, sequence and acknowledgment numbers, and the Control Bits (URG: Urgent Pointer field significant, ACK: Acknowledgment field significant, PSH: Push Function, RST: Reset the connection, SYN: Synchronize sequence numbers, FIN: No more data from sender).

  To establish a connection between two hosts, the TCP three-way handshake is used which works as follows (C refers to the client and S refers to the server):

  1. C sends a SYN packet to S
  2. S sends a SYN/ACK packet to C
  3. C sends an ACK packet to S

  To terminate a previously established connection, the four-way handshake is used which works as follows (C refers to the client and S refers to the server):

  1. C sends a FIN packet to S
  2. S sends an ACK packet to C
  3. S sends a FIN packet to C
  4. C sends an ACK packet to S

- **UDP:** The User Datagram Protocol is a connectionless protocol. UDP takes the data an application provides, packs it into a UDP packet, and hands it to the Network layer. The packet is then put on the wire, and sent to the other end. There is no way to guarantee that the packet will reach its destination. Important fields in the UDP header are: the source and destination port numbers which identify services.

- **ICMP:** The Internet Control Message Protocol is mainly used for managing and controlling IP. With it, you can force routing options, declare a destination unreachable, etc. The most widely used functions, however, are echo-request and echo-reply. These two functions are used to see if you can reach a certain host on your network. Important fields in the ICMP header: the message type (e.g. echo-request, echo-reply, time-exceeded in transit, etc).

Implementing a network protocol or application requires an API to work with. The sockets API is by far the most widely used networking API. A socket is one end-point of a two-way communication link between two programs running on the network. One type of sockets which are known as raw sockets are of particular interest to attackers. Raw sockets offer the programmer absolute control over the data which is being sent or received through the network. In other words, the programmer gets full control over every single bit which is sent to a network. Skilled attackers build their own crafted packets that can bypass firewalls and intrusion detection systems, hijack sessions, cause systems to crash, consume bandwidth, etc.

2.2 Cryptography

Cryptography is the study of how to make information undecipherable except to the people that you wish to understand it. Cryptography comes from the Greek words for "secret writing" and it has a long and colorful history going back thousands of years [11]. The basic idea of applying any cryptographic technique is simple. Consider a sender S wanting to transmit message M to a receiver R. To protect M from passive and active attacks, the sender S can encrypt M into an unintelligible message C and send it to R. Upon receiving M, R decrypts C into its original form M. Encryption and decryption are done using cryptographic techniques parameterized by keys. The original form of the message before encryption is called plaintext and after encryption it is called ciphertext.

A cryptosystem where encryption and decryption of a message M are done using the same key is known as a symmetric cryptosystem. On the other hand, if the cryptosystem uses two different keys for encryption and decryption then it is known as an asymmetric cryptosystem. Examples for symmetric cryptosystems would be DES (Data Encryption Standard) and AES (Advanced Encryption Standard) and for asymmetric cryptosystems would be RSA, and ELGamal.

One important application of cryptography is the use of hash functions. A hash function H takes a message M of arbitrary length as input and outputs a fixed sized bit string. Hash functions are one-way functions which means that for any given M, it is easy to compute $h = H(M)$ where h is the hash value. However, it is computationally infeasible to find an M such that $M = H^{-1}(h)$

Cryptographic techniques and applications are used to provide data confidentiality, data integrity, nonrepudiation, authentication, and many more. Such techniques have been used in many network protocols to provide better security services and mechanisms (e.g. SSH, SSL).

2.3 Firewalls

In the context of buildings, a firewall is a fireproof wall intended to prevent the spread of fire from one room or area of a building to another. It has acquired a related meaning in the context of the Internet. A typical intranet these days is not connected to the Internet directly. Instead, the intranet is connected to a firewall, and the firewall is connected to the Internet. A firewall is a computer system dedicated to protect a LAN

(Local Area Network) from the Internet. It is usually placed at the entry point of the LAN it protects and it can be simple packet filters to an enormously complex computer systems with extensive logging systems, intrusion detection systems, etc.

A personal firewall is a software that is installed on a single system which receives, inspects and makes decisions about all incoming and outgoing traffic from the LAN to the system and vice versa.

2.4 Intrusion Detection Systems

Building an intrusion prevention system that can prevent all malicious attacks is a desired but, at the same time, an impossible task. Any intrusion prevention system will fail at one point and hence the second line of defense for any system is to use an intrusion detection system which can be used to identify the attacker and strengthen the intrusion prevention facility [11].

Intrusion detection is the art of identifying attempted or ongoing attacks on a computer system or network. There are two approaches that intrusion detection technologies use:

1. **Statistical Anomaly Detection:** This type of detection involves the collection of data relating to the behavior of legitimate users over a period of time. The statistical tests are then applied to observed behavior to determine with a high probability whether the behavior is not legitimate user behavior. Such type of detection can be further divided into two types:
   - **Threshold detection:** This approach involves defining thresholds, independent of user, for the frequency of occurrences of various events [11].
   - **Profile based:** A profile of the activity of each user is developed and used to detect changes in the behavior of individual accounts [11].
2. **Rule-Based Detection:** This type of detection involves defining a set of rules that can be used to decide that a given behavior is that of an intruder. Such type of detection can be further divided into two types:
   - **Anomaly Detection:** Rules are developed to detect deviation from previous usage patterns [4].
   - **Penetration identification:** An expert system approach that searches for suspicious behaviour in network traffic[8].

Strengths of Intrusion Detection Systems:

- Provides worthwhile information about malicious network traffic.
- Can be programmed to minimize damage.
- Help identify the source of the incoming probes or attacks.
- Can collect forensic evidence, which can be used to identify intruders.
- Alerts security personnel that a network invasion may be in progress.

Weaknesses of Intrusion Detection Systems:

- Produces false positive (false alarms).

- Produces false negative (failed to alarm).
- Large-scale attacks could overwhelm a sensor.

2.5 Attackers

The first line of defense against malware is to know who are the people responsible for writing or releasing malicious code and what motivates them. A longstanding debate exists in the computer security field about the correct terminology to use to describe an attacker [1]. Due to the misuse of the word hacker by the media, the public always refer to people penetrating systems as hackers. However, at one point in time, a hacker was someone who is obsessed by programming, someone who programs quickly and enjoys stretching the capacity of operating systems as opposed to a normal user who will be satisfied by knowing the minimum that will get the job done. An example of real hackers would be the people who wrote UNIX and who were called at that time the UNIX hackers. To clarify the true picture, a list of the most common terms used in the computer security field to describe attackers are as follows:

- **Script Kiddies:** This group of people call themselves real hackers but in fact some are novice programmers and others are not even programmers at all. A script kiddie searches the Internet for exploits (malicious code) written by highly skilled programmers and use these exploits to break into systems. Many of the attacks these days are caused by script kiddies who release malicious code and wreak havoc. An example for such acts would be the release of Code Red II which was an identical copy of Code Red I except for one string in the original worm's code that was changed to a different string. It is indeed obvious that script kiddies motivation is fame.
- **Crackers:** This group of people are known as average attackers who have medium-level programming skills. Crackers usually use the same tools used by script kiddies. However, crackers do know how the code of such tools is written and how it behaves. Crackers do not have any ethical boundaries and their motivations behind mounting attacks against systems and networks is money and power.
- **White Hat Hackers:** This group of people are highly skilled programmers who use their skills in enhancing the security of software. White hat hackers work in the computer security field and they are indeed known to be the best developers.
- **Black Hat Hackers:** The technical skills of black hat hackers are at a level comparable to that of white hat hackers [1]. However, the focus is different. White hat hackers are more concerned about the security issues of software development and algorithms. On the other hand, Black hat hackers are more concerned about the security of systems.

3 Network Attacks and Countermeasures

It has been long known that those who know how their system can be exploited are far superior at preventing such attacks than those who do not. Thousands of attacks exists

these days and the description of every single attack is beyond the scope of this paper. However, each attack simply starts with a simple set of steps and most of the time with similar techniques. This section gives a detailed description of the most used attacks, along with some analysis of such attacks and techniques for better countermeasures. A number of programs have been used to simulate some of the attacks and the generated packets were captured using a packet sniffer called tcpdump. The IP addresses in the captured traces have been modified for illustration purposes.

3.1 Packet Sniffing

Packet sniffing is a passive attack where packets are read from a network stream. A packet sniffer is a piece of software or hardware that monitors all network traffic. Tools like tcpdump/windump, Ethereal, and many more are examples of packet sniffer softwares that use raw sockets and the recvfrom system call to read packets from the network interface. These tools were originally developed to allow network administrators to troubleshoot their network and find bottlenecks, and by Intrusion Detection Systems to monitor for malicious traffic. Anybody with root privileges on Linux or administrator privileges on Windows can run a packet sniffer and start capturing packets. The security threat presented by sniffers is their ability to capture all incoming and outgoing traffic, including usernames, passwords, and other sensitive information that travels over the network in the clear.

To demonstrate the threat of such tools, we can look at telnet for instance. Telnet is a protocol that does not use any encryption schemes and all the data gets transmitted in the clear between the two communications parties. If user A uses the telnet protocol to connect to server B, then user C who is running a packet sniffer between A and B can see the plaintext traffic. C can capture the username and password of A, use it to break into A's account on machine B, and attempt to gain access to a higher privileged account (i.e. root/administrator). User C can be more imaginative and install a sniffer on the server to compromise more and more accounts or even use the compromised accounts to mount a more sophisticated attack.

Example:
Here is an example of a sniffer capturing a TCP three-way handshake between machine A and machine B to start a telnet session:
IP A.33819 > B.telnet: S 2797576379:2797576379(0)
IP B.telnet > A.33819: S 2802411635:2802411635(0) ack 2797576380
IP A.33819 > B.telnet: . ack 1

3.1.1 Countermeasures

Packet sniffing is a passive threat that does not alter the packets that are being transmitted. It is, therefore, quite difficult to know whether a specific system is running a packet sniffer or not. However, some techniques can be used to determine whether the NIC (Network Interface Card) on the suspect machine is running in promiscuous mode or not. The non-trusting assumption is that because it is in promiscuous mode,

the machine must be running a sniffer. These tests assume that the sniffer detection tool and the suspect machine are both in the same Ethernet segment:

- **DNS test:** Create numerous fake TCP connections expecting a poorly written sniffer to pick up on those connections and resolve the IP addresses of the nonexistent hosts. Some packet sniffers perform reverse DNS lookups for the packets it captures. When reverse DNS lookup occurs, the detection tool sniffs the lookup request to see if the target is the one requesting resolution of that nonexistent host [13].
- **Ping test:** This method relies on a problem in the target machine's kernel. One can build an ICMP echo request packet with the IP address of the machine suspected of hosting a sniffer but with a deliberately mismatched MAC address [13]. Send the ICMP packet to the target with the correct destination IP address, but a bogus destination hardware address. Most systems will disregard this packet since its hardware address information is incorrect. But in some Linux, NetBSD and NT systems, since the NIC is in promiscuous mode, the sniffer will grab this packet off the network as a legitimate packet and respond accordingly. If the target in question replies to our request, we know it is in promiscuous mode. Clever attackers are of course aware of this and can update their sniffers to filter out such packets.

The above tests can be used to help detect machines that are running sniffers. To prevent such threats, the best solution is to use protocols that use strong cryptographic schemes to encrypt all the incoming and outgoing traffic.

3.2 Ping and Traceroute

Before the attacker attempts to break into a system or a network, some information about the target needs to be gathered first. Knowing the topology of any network and which systems are up or down at any given time is considered knowing a lot of sensitive information. Two programs, traceroute and ping, were initially developed to allow network administrators to troubleshoot their networks by sending appropriate probes. Both ping and traceroute make use of the ICMP protocol to accomplish their tasks. The abuse of such programs in general and the ICMP protocol in particular helped many attackers to gather sensitive information about their targets and sometimes cause a denial of service attack as described in the Denial of Service section. A user with root or administrator privileges can run the traceroute or ping programs, or even construct ICMP packets and sends such packets to hosts on the Internet.

3.2.1 Ping

The ping program is used to determine whether machine B is up or down at any given time t. The way it works is as follows (Assume that the ping program is running on machine A):

1. sets the message type in the ICMP header to an echo request message, the source address in the IP header to A's IP address, and the destination address in the IP header to B's IP address. The ICMP packet is then sent to B.

2. If B receives the ICMP echo request message then it will send A an ICMP echo reply message.
3. If A receives an ICMP echo reply then B is considered to be up at time t. Otherwise, it is considered to be down.

Ping Trace:
- The attacker sends the victim an ICMP echo request:
IP attacker > victim: icmp 64: echo request
- The victim sends the attacker an ICMP echo reply:
IP victim > attacker: icmp 64: echo reply

3.2.2 Traceroute

The traceroute program is used to count how many routers are between machine A and B. The Unix version of the traceroute program uses two different techniques (ICMP by itself or with UDP). The Windows version uses only ICMP. The way such programs work is as follows:

- **ICMP:**
  1. Sets the message type in the ICMP header field to an echo request message, the source address in the IP header to A's IP address, the destination address in the IP header to B's IP address, and the TTL value in the IP header to 1. The ICMP packet is then sent to B.
  2. If a router receives the packet, then the TTL value is decremented by one. If the router decrements and gets a 0 then the packet is discarded and the router constructs an ICMP packet A with a time exceeded in transit message. Otherwise, the packet is forwarded to the next router or to B
  3. If the B receives the ICMP echo request message then it will send back an ICMP echo reply message.
  4. If the traceroute program on A gets a time exceeded in transit message then step 1 is repeated but with a TTL value equal to 2 (Note that the TTL value keeps incrementing until an echo reply is received).
  5. If the traceroute program on A gets an ICMP echo reply from B then the number of routers between A and B is simply the current value of TTL - 1.
- **UDP and ICMP:** The same process is repeated as described above but with a few small differences:
  1. The program sends a UDP packet to a random udp port on B instead of an ICMP packet.
  2. A expects to receive an ICMP port unreachable message from B instead of an echo reply.
  3. Similarly to above, the number of routers between A and B is simply the current value of TTL - 1.

Traceroute ICMP Trace:
- The attacker sends the victim an ICMP echo request:

IP attacker > victim: icmp 18: echo request
- The victim sends the attacker an ICMP echo reply:
IP victim > attacker: icmp 18: echo reply
Traceroute UDP Trace:
- The attacker sends the victim a UDP packet from udp port 32840 to udp port 33435 :
IP attacker.32840 > victim.33435: UDP, length: 10
- The victim sends the attacker an ICMP udp port unreachable:
IP victim > attacker: icmp 46: victim udp port 33435 unreachable

3.2.3 Countermeasures

The best way to prevent against malicious ICMP probes is to block all ICMP packets. To prevent against incoming and outgoing ICMP packets from the Internet to a LAN and vice versa, a firewall protecting the LAN can be configured to block such packets. To prevent probes from machines to other machines on the same LAN, either a router can be configured to not send any ICMP packets or a personal firewall can be installed and configured to block the packets. One should take into consideration that blocking all ICMP packets is not the optimal solution because such packets are also used for non-malicious purposes. Consider the following scenario for instance: if machine *A* wants to send some data to machine *B* on some LAN. *A* sends a huge packet which exceeds the MTU of *B*'s LAN. Before the packet enters the LAN, it must pass through a router and if the router is allowed to send ICMP packets then it will simply discard the packet and send an ICMP packet with a "need to frag" message along with the LAN's MTU. *A* receives the ICMP packet and knows that it must split the packet into fragments and send them to *B*. What if the router has been silenced and configured not to send any ICMP packets? If that is the case, then *A* will never know that the packet was discarded.

3.3 IP spoofing

IP spoofing is a technique that is used to send packets with a spoofed IP address. Attackers attempt to use such techniques to hide their identities and make it difficult for authorities to track the source of an attack. The IP header contains a 32-bit field for the source IP address and a 32-bit field for the destination IP address. One can simply construct a packet and fill the source address in the IP header with any IP address (either valid or invalid). Once the packet is constructed then the attacker can call the sendto system call to send the packet. For instance, nmap (a port scanner) which constructs crafted packets can be used to spoof the source of a scan using the following command (must have root privilege):
nmap -S spoofed-address -e eth0 -sP target-address
The above command will construct an ICMP echo request packet, sets the source address field in the IP header with the *spoofed-address* and sends the packet to *target-address*. Using the above command in this case is indeed useless because if the *target-address* is up then it will send an ICMP echo reply to *spoofed-address* and the attacker will never know whether the target is up or down. However, the above command

illustrates how the source of any packet can be easily spoofed. IP spoofing is mostly used when attackers mount denial of service or distributed denial of service attacks to cause such attack to work effectively (refer to the Denial of Service and Distributed Denial of Service sections for more details).

3.3.1 Countermeasures

If the spoofed address is for instance 127.0.0.1 or one of the IP addresses that are used for experimental or private purposes then one can detect such spoofing easily. Otherwise, it is quite difficult to know whether an address is spoofed or not. Consider for instance the following trace:
- The attacker sends the victim an echo request:
IP spoofed-address > victim: icmp 64: echo request
- The victim sends the attacker an echo reply:
IP victim > spoofed-address: icmp 64: echo reply

If the above trace was captured, then one might think that *spoofed-address* attempted to ping *victim.* is it possible, however, to know whether the source IP address is a real IP or a spoofed one? The answer to this question is it depends. To be able to know whether a spoofed address was used or not, one needs to analyze more traces which shows the given address and its related activities. Even then determining whether the used IP address is real or spoofed is still not guaranteed.

3.4 Port Scanning

Once the attacker knows that the victim's machine is up then the second step is to know what services the system is running. Some system and network administrators simply ignore port scanning activities against their systems and networks because they think that no harm can be caused by such activities. One should know, however, that if someone scanned your network or system then it must have been done with malicious intents otherwise the scanning activity would not have been done in the first place. If a thief wants to break into a house, then first the thief will start checking whether the doors/windows are open or closed, whether somebody is inside the house or nobody is home, etc. One can think of port scanning as exactly what a clever thief would do before breaking into a house. Many different scanning techniques exist today, some are stealthier than others and can bypass firewalls and Intrusion Detection Systems easily. If strict countermeasures are applied against port scanning activities then attackers would have to think twice before attempting to scan a system or a network. Many port scanning tools can be found on the Internet but the most famous port scanner is called nmap. Nmap requires root or administrator privileges so it can construct crafted packets and sends them. A list of the most famous and stealthy scans used by nmap and many other port scanners are described below:

3.4.1 TCP connect scan

This is the easiest type of scanning techniques since it does not require root or administrator privileges. Here is how it works (A refers to the attacker and B refers to victim's machine):

1. Construct a TCP stream socket and call the connect system call offered by the kernel.
2. If the system call return -1 (indication of an error of some sort) then the port is closed. Otherwise, it is considered open.

TCP connect scan trace:
The attacker attempts a TCP connect scan to an open port 22 (ssh) from port 34401. The TCP three-way handshake is completed and because of that everything gets logged on the victim's system (this is why this type of scan is not considered stealthy).
- The attacker sends a SYN to the victim:
IP attacker.34401 > victim.ssh: S 114661504:114661504(0)
- The victim sends the attacker a SYN/ACK:
IP victim.ssh > attacker.34401: S 102340587:102340587(0) ack 114661505
- The attacker sends the victim an ACK:
IP attacker.34401 > victim.ssh: . ack 1

The attacker attempts a TCP connect scan trace to a closed port 31347 from port 34405. It is obvious that this type of scan can only be stealthy if it is mounted against a closed port since no full connection takes place.
- The attacker sends the victim a SYN:
IP attacker.34405 > victim.31347: S 949824287:949824287(0)
- The victim sends the attacker a RST/ACK:
IP victim.31347 > attacker.34405: R 0:0(0) ack 949824288

3.4.2 SYN scan

This type of scan is known as half-open scanning because the scanner does not finish the three-way TCP handshake. Here is how it works (A is the attacker and B is the victim's machine):

1. A sends a SYN packet to a specific port on B.
2. If the port is closed then B sends a RST/ACK packet to A, else if the port is open then B sends a SYN/ACK packet to A.
3. If A receives a RST/ACK packet then A knows that the port is closed, else if A receives a SYN/ACK packet then A knows that the port is open and sends back a RST packet to tear down the connection.

SYN scan trace:
The attacker attempts a SYN scan to an open port 22 (ssh) from port 46813 and only the first two steps of the TCP three-way handshake is completed (nothing is logged on the victim's system and this is why this type of scan is considered stealthy).
- The attacker sends the victim a SYN:

IP attacker.46813 > victim.ssh: S 3098388824:3098388824(0)
- The victim sends the attacker a SYN/ACK:
IP victim.ssh > attacker.46813: S 1752551723:1752551723(0) ack 3098388825
- The attacker sends the victim a RST:
IP attacker.46813 > victim.ssh: R 3098388825:3098388825(0)

The attacker attempts a SYN scan to a closed port 31347 from port 39359
- The attacker sends the victim a SYN:
IP attacker.39359 > victim.31347: S 1717193123:1717193123(0)
- The victim sends the attacker a RST/ACK:
IP victim.31347 > attacker.39359: R 0:0(0) ack 1717193124

3.4.3 FIN scan

Many firewalls block incoming SYN packets to restricted ports and to bypass such devices and packet filters, an attacker can use a FIN scan. Here is how it works (A is the attacker and B is the victim's machine):

1. A sends a FIN packet to a specific port on B.
2. If the port is closed then B sends a RST/ACK packet to A, else if the port is open then B does not send anything to A
3. If A receives a RST/ACK packet then A knows that the port is closed, else if A does not receive any response from B and it times out then the port is open.

FIN scan trace:
The attacker attempts a FIN scan to an open port 22 (ssh) from port 54181.
- The attacker sends the victim a FIN:
IP attacker.54181 > victim.ssh: F 0:0(0)
- The attacker does not receive any response and hence the ssh port is considered to be listening.

The attacker attempts a FIN scan to a closed port 31347 from port 57955.
- The attacker sends the victim a FIN:
IP attacker.57955 > victim.31347: F 0:0(0)
- The victim sends the attacker a RST/ACK:
IP victim.31347 > attacker.57955: R 0:0(0) ack 1

3.4.4 NULL and XMAS tree scans

These type of scans can be used to both determine open/closed ports and fingerprint the victim's operating system. In the XMAS tree scan, the attacker sends a packet to the victim's machine with the following flags set: URG/PSH/FIN. If the port is closed then a RST/ACK packet is sent back. Otherwise, the port is considered open. In a NULL scan, the attacker sends a packet without setting any flag and again if a RST/ACK packet is sent back then the port is considered closed and otherwise open.
NULL scan trace:
The attacker attempts a NULL scan to an open port 22 (ssh) from port 61705.

- The attacker sends the victim a packet without setting any flag:
IP attacker.61705 > victim.ssh: .
- The attacker does not receive any response, times out and the ssh port is considered to be listening.

The attacker attempts a NULL scan to a closed port 31347 from port 57367.
- The attacker sends the victim a packet without setting any flag:
IP attacker.57367 > victim.31347: .
- The victim sends the attacker a RST/ACK:
IP victim.31347 > attacker.57367: R 0:0(0) ack 0

XMAS tree scan trace:
The attacker attempts an XMAS tree scan to an open port 22 (ssh) from port 60836.
- The attacker sends the victim a FIN/PSH/URG:
IP attacker.60836 > victim.ssh: FP 0:0(0) urg 0
- The attacker does not receive any response and hence the ssh port is considered to be listening.

The attacker attempts an XMAS tree scan to a closed port 31347 from port 58129.
- The attacker sends the victim a FIN/PSH/URG:
IP attacker.58129 > victim.31347: FP 0:0(0) urg 0
- The victim sends the attacker a RST/ACK:
IP victim.31347 > attacker.58129: R 0:0(0) ack 1

3.4.5 Countermeasures

To detect port scanning activities, here are some ideas:

- If the system logs a full TCP connection and no data was exchanged then its probably a port scan.
- An effective way to detect most of the stealthy scans is to monitor both incoming and outgoing traffic. If your machine is sending out a RST/ACK packet then it means that somebody tried to connect to a closed port on your machine. What will make somebody do that? Sometimes its probably a coincidence but most of the time it implies that someone is scanning your system.

To protect against port scanning activities coming from the Internet to a LAN, then a firewall can be configured to filter and only allow packets coming to open ports only and not any port.

3.5 Denial Of Service

Denial of Service (DoS) is an attack that prevents a system or a network from using its resources normally. These attacks are known to abuse Internet citizens, to stop network traffic and commerce to corporate websites, and to suppress the network presence of hosts that an attacker wants to impersonate. IP spoofing is a common technique that is used when attackers mount denial of service attacks to hide their identities and/or

to cause their attacks to succeed. A list of the most common DoS attacks are described in this section:

3.5.1 SYN flood

This attack exploits the TCP three-way handshake and causes the victim's machine to crash. In this attack, the attacker sends a large number of SYN packets to the victim. The success of this attack depends on the number of packets sent and the speed with which they are sent. To make the attack work effectively, attackers usually fork many child processes that can send many packets to the victim at the same time. A SYN flooder engine works as follows (A refers to the attacker and B refers to the victim's machine):

1. A constructs a TCP packet and does the following: sets the SYN flag in the TCP header, fills the source address in the IP header with a spoofed IP address (the source IP address must be a machine that is down to get this attack to work), fills the destination address in the IP header with B's IP address, and finally fills the destination port number in the TCP header with a port on B that is listening and accepting connections.
2. A calls the sendto system call and the packet is sent to B.
3. B receives a SYN packet and since the port on B is accepting connections then B adds the packet to a queue of half-open connections. B will attempt to finish the three-way handshake and as a result a SYN/ACK packet is sent to the spoofed IP address in the packet. Since the spoofed IP address is a machine that is down (i.e. not connected to the Internet at this time), then no response will be sent back to B but B will keep waiting for an ACK packet.
4. A repeats the same procedure in step 1 and sends more and more SYN packets until the queue of half-open connections on B gets filled up (the queue will always have a fixed size because memory is fixed). Once B's queue gets filled, then B will usually crash.

SYN Flood trace
The attackers attempts to flood an open port 22 (ssh) with SYN packets, and uses a spoofed (unused) IP address. For illustration, only 5 packets have been taken from the captured session.
IP unused-address_1.34566 > victim.ssh: S 674719801:674719801(0)
IP victim.ssh > unused-address_1.34566: S 2595256625:2595256625(0) ack 674719802
IP unused-address_2.34566 > victim.ssh: S 674719801:674719801(0)
IP victim.ssh > unused-address_2.34566: S 2708979260:2708979260(0) ack 674719802
IP unused-address_3.34566 > victim.ssh: S 674719801:674719801(0)
IP victim.ssh > unused-address_3.34566: S 2617688745:2617688745(0) ack 674719802
IP unused-address_4.34566 > victim.ssh: S 674719801:674719801(0)
IP victim.ssh > unused-address_4.34566: S 2629081320:2629081320(0) ack 674719802
IP unused-address_5.34566 > victim.ssh: S 674719801:674719801(0)
IP victim.ssh > unused-address_5.34566: S 2631113410:2631113410(0) ack 674719802

3.5.2 Ping of Death

The ICMP protocol is used to mount this attack. Some software allows the attacker to send ICMP packets that are larger than 65,536 bytes which is the maximum that the TCP/IP specification allows [7]. Since the packet is large, then it gets fragmented to smaller packets and sent to the victim's machine. Once the victim receives all the fragments and reassembles them to their illegal size, the buffer that stores the complete packet overflows and causes the system to crash.

3.5.3 Countermeasures

Prevention techniques are described below:

- SYN Flood: A very simple way to prevent against this attack can be done by adding appropriate checks to the queue of half-open connections. If the queue is about to get filled up then abort all the connections and flush the queue.
- Ping of Death: Keep track of the current fragment size and next fragment size and if adding the next fragment size exceeds 65,536 bytes then discard the packet.

3.6 Distributed Denial of Service

Distributed Denial of Service (DDoS) attacks are a more sophisticated way to flood machines and exhaust their resources. It is based on the same idea of DoS but with a small variation in mounting the attack which can make it more effective in hitting the target and forcing it to reboot. DDoS are of big concern to system administrators and security analysts because the attacks are done from compromised hosts on the Internet and tracking down the original source of the attack is extremely difficult. Two terms: zombie, and handler are used when one talks about DDoS. These two terms are described below:

- **Zombie:** A Zombie refers to a compromised host on the Internet. A host can be compromised using one or more of the following techniques: sniffing plaintext username and passwords, sniffing encrypted username and passwords and cracking the encryption scheme, obtaining a copy of the password file and cracking the encrypted passwords, social engineering, exploiting a software bug on the host machine, etc. If the attacker can break into a host, then the attacker can install a backdoor to access the system again (i.e. breaking into a system and adding a new user account, installing a Trojan horse, etc).
- **Handler:** A handler refers to a program that communicates with another program installed on a Zombie to cause it to execute specific commands according to what the master wants (the attacker).

A generic scenario to mount a DDoS can be achieved as follows (A refers to the attacker and B refers to the victim's machine):

1. A writes a program called *syn-attack* with a SYN flooder engine that can send SYN packets to B with a spoofed and unused source IP address. A adds some code to cause the *syn-attack* program to listen to tcp port 41414 once executed

and to wait for an incoming string "Attack Target". If the malicious program receives the string then it will start executing the SYN flooder's code. *A* injects the *syn-attack*'s code into an executable file of a program that is used very often by many users call it *org-program*. *A* can simply append the malicious code to the end of the executable file and modify the structure of the file to allow the code to get loaded into memory along with the original program and starts executing before the original program. Once the malicious code gets executed and becomes memory resident then it simply passes the control back to the original program.

2. *A* breaks into a number of hosts on the Internet and replaces the *org-program* on the compromised host with the modified version which is the same program but contains an embedded malicious code.
3. *A* writes the handler's code which once executed attempts to connect to the IP addresses of the compromised hosts on port 41414 and sends the string "Attack Target".
4. Users on the compromised hosts execute *org-program* and the malicious code starts listening on tcp port 41414.
5. *A* executes the handler's code and the string is sent to the zombies.
6. Ports 41414 on the compromised hosts receive the string and start sending a large number of SYN packets to *B* and *B* gets flooded.

The above scenario is a generic simple way to mount a DDoS attack. Many different techniques can be added by attackers to allow a more sophisticated and successful attack.

3.6.1 Countermeasures

DDoS attacks rely on the zombies to mount the attack. If users use strong password and keep their operating system and software up to date then it will be difficult for attackers to break into their systems. If the hosts are compromised then one can use some of the techniques described in the DoS countermeasure section.

3.7 Buffer Overflow

Any program uses a buffer at one point or another. A program which uses a fixed size buffer and does not check the buffer's boundry then the program is vulnerable to a buffer overflow attack. Many libc functions in C/C++ do not check buffer boundaries (i.e. strcpy, gets, strcat, etc) and by using such functions in any program, attackers can execute arbitrary code and change the flow of the vulnerable program. Buffer overflow bugs are the most significant practical security threat of the decade and many worms exploit such bugs to gain root or administrator privilege on the victim's machine. There are many different types of buffer overflow attacks: stack smashing, heap smashing, return-into-libc, etc. For illustration, the simplest form of such attacks which is known as stack smashing will be described in this section.

- Whenever a function is called then the arguments of the function are pushed onto the stack, then the return address of where the function is called (An absolute

memory address) which is 4 bytes, and finally the saved frame pointer which is used to access the local variables of the function which is 4 bytes.

- Lets assume that a buffer called buf is declared inside a function and has a fixed size say 1024 bytes. For simplicity, we will assume that all that the function does is to read from standard input and echo the input back to standard output once its done reading (i.e. if a user hits enter then function knows its done reading).
- If a user executes this program and enters data less than the buffer's fixed size then nothing will happen and the program will execute normally. However, If a user enters more than the fixed size value plus 5 bytes to overwrite the saved frame pointer and the first byte of the return address then a segmentation fault will occur.

The hardest part in buffer overflow attacks is to know the buffer's address on the stack. To get an approximation of the size of the buffer, one can keep inputting data to the vulnerable program and count the number of bytes used until a segmentation fault occurs. If the attacker is able to get the stack pointer address once the function is called then the attacker can subtract the approximate size from the stack pointer and gets an approximations of the buffer's address. Debugging techniques can be used to find out the stack pointer address if the executable of the vulnerable program is available. To be able to exploit the bug in the above program, an attacker can do the following:

- Write the payload in assembly or simply use C and then convert it to assembly code with the -S flag.
- Optimize the assembly code and get rid of any end of string characters which is used by many libc functions to indicate the end of the string. For instance if an assembly instruction sets a register to 0, then an optimization would be to xor the contents of the register with itself.
- Convert the assembly instructions to its hexadecimal representation using a debugger. If the hexadecimal exploit string is less than 1024 which is the size of the buffer in the above program then one can simply add nop instructions to the exploit string. Say the payload string is 500 bytes then a candidate string to smash the stack and run the payload can be crafted as follows:
  528 bytes (nop instructions) + 500 bytes (exploit string) + 4 bytes (starting address of the buffer on the stack)
  The above crafted exploit will overflow the buffer and overwrites the return address with the buffer's address. Once the code of the function is done executing, then the buffer address will be fetched and the program counter will point to it and start executing the nop instructions followed by the exploit code.

3.7.1 Countermeasures

Some techniques like monitoring the return address on the stack, making the stack non-executable, etc have been developed to help protect against buffer overflow. However, an effective countermeasure for this type of attack has not been implemented. Keeping

your operating system and software up to date can sometimes help. The best solution, however, would be to write secure code in the first place by checking buffers size limits.

3.8 Trojan horses

A Trojan horse is a malicious program which is embedded inside an innocent program (i.e. a game). Such programs are based on a client/server model where the server program gets installed on the victim's machine and the attacker uses the client program to send commands to the server, hence giving the attacker complete control over the victim's machine. A generic way to build a simple Trojan horse that can for instance copy random or specific files from the victim's machine and send it to the attacker is as follows (A is the attacker and B is the victim's machine):

1. A writes the server code of the Trojan horse that once executed does the following: creates a socket and binds it to tcp port 56565, listens for incoming connections to that port and waits for a specific string "Get file" where file refers to the file name that is to be copied. Once the server receives such string then it starts traversing the file system searching for the file. Once the file is found then the server's code starts copying and sending the bytes of the file to the client program running on the A's machine. A can inject the server's code into an executable file of for instance a game and fool the victim to execute it. A will have to modify the structure of the game's executable file to get the malicious code to run first and then the game.
2. A writes the client code of the Trojan horse which once executed does the following: creates a socket, connects to the IP address of B port 56565, sends the string "Get file" and waits for the copy of the file.
3. A sends the game to the victim.
4. The victim executes the game without knowing that it is embedded with malicious code. The server's code starts running and goes memory resident and then the game's code starts to execute.
5. A runs the client code and gets a copy of whatever file A wants

Many different techniques can be added and used by Trojan horses to allow the attacker to send more sophisticated commands remotely and by using better features for instance to know when the victim is up, and so on.

3.8.1 Countermeasures

Do not download and execute code from untrusted websites. Keeping your anti-virus software up to date can help detect known Trojans. However, anti-virus software will not be able to detect Trojans whose code can be found in the wild (the Internet).

3.9 Viruses and Worms

Viruses and Worms are by far the most dangerous types of Malware. Such malicious code can exhaust resources, invade privacy, and cause millions of dollars in losses.

What makes viruses and worms a unique piece of code is their ability to replicate. Worms are independent programs that can be executed on the CPU like any other program. However, viruses need a host program and they only can execute once the host program is executed. Any malicious program can be written in such a way that it behaves like a Trojan, a virus, a worm, or a combination of the three. Worms are more network related and once they are released, a lot of bandwidth gets consumed due to the heavy traffic caused by their propagation phase. To fully understand how a worm propagates, some assumptions need to be made first:

- Assume that on a given network, half of hosts are running a service called *example* which listens to tcp port 4444 and has a buffer overflow bug that allows a remote attacker to execute arbitrary code.
- Assume that an attacker writes a worm which exploits the *example* service and which uses a hit list scanning technique where the list of IP addresses of the vulnerable machines running the *example* service is included within the worm's code (To determine the vulnerable machines, the attacker can simply use port scanning techniques).

The attacker releases the worm by for instance sending it to machine E. To demonstrate the worm's attack, here is a simple scenario:

1. A user on machine E executes the worm's code without knowing what it does.
2. The worm's code (the parent) is now running on E's CPU. The worm opens a tcp port 23456 on E, forks a child process to wait for incoming connections on that port and to start sending a copy of worm's code to any machine that attempts to connect. The parent process continues executing on E's machine and selects a random IP address from the list that was added in its code by the attacker. Lets say the selected IP address is machine F's IP address.
3. F is one of the vulnerable machines that are running the *example* service. Therefore, once the parent worm selects F's IP address then it attempts to connect to tcp port 4444, patch the exploit code on the fly with E's IP address, and sends the exploit to the *example* service running on F.
4. The *example* service on F overflows and the exploit code gets executed and does the following: connects to E's IP address on tcp port 23456 and starts copying the worm's code from E to F and executes it.
5. Go to step 2 but with machine F instead of E.

3.9.1 Countermeasures

Effective countermeasure against worms is still not yet found. However, keeping your operating system, softwares and anti-virus software up to date will help protect your system to some extent.

4 Experiments

Snort, an intrusion detection system, was used along with IDPP (a system that was implemented to accelarate the detection and analysis of attacks) to analyze intrusion attempts in real data sets captured at the University of Calgary. Snort is a lightweight network intrusion detection system, capable of performing real-time traffic analysis and packet logging on IP networks. It can perform protocol analysis, content searching/matching and can be used to detect a variety of attacks and probes, such as buffer overflows, stealth port scans, CGI attacks, SMB probes, OS fingerprinting attempts, and much more. Snort uses a flexible rules language to describe traffic that it should collect or pass, as well as a detection engine that utilizes a modular plugin architecture. Snort has a real-time alerting capability as well, incorporating alerting mechanisms for syslog, a user specified file, a UNIX socket, or WinPopup messages to Windows clients using Samba's smbclient [9]. For this project, Snort was installed and configured to process trace files captured at the University of Calgary and detect intrusion attempts.

IDPP which stands for Intrusion Detection Post-Processing was implemented in Java. The system reads the alert file generated by Snort, parses it and dumps the new collected data to a MySQL database. The system has a Graphical User Interface to make the visualization of attack more effective, and uses a MySQL database to effectively allow the user to browse and query a huge amount of data in a reasonable time. The main focus of the implementation was to provide a more readable format of the intrusion attempts, to get some statistics about the intrusions, and to provide better ways to quickly search for specific attacks. The system contains four main classes:

- **Parser:** This class is used to perform the parsing routines.
- **GUI:** This class is used to build the Graphical User Interface and perform the interaction routines.
- **Database:** This class is used to perform the database routines.
- **Mail:** This class is used to send mail using SMTP (Simple Mail Transfer Protocol).
- **Load file:** This option allows the user to load an alert file generated by Snort. Once the file is selected, it gets parsed by the parser, and gets loaded to both a MySQL database table and to the Graphical User Interface.
- **Get file statistics:** This option allows the user to get some statistics about the alert file. This includes the total number of packets, the number of TCP packets, the number of UDP packets, and the number of ICMP packets.
- **Get attack statistics:** This option allows the user to get some statistics about the attacks in the alert file. It displays uniquely the attacks found in the file and how many times each of them occured.
- **Get U of C attack statistics:** This option allows the user to get statistics about how many attacks in total were mounted from the University of Calgary's network to other networks and vice versa.

- **Get all attack statistics:** This option allows the user to get statistics about how many attacks were mounted from the University of Calgary's network to other networks for all the attacks found in the alert files and vice versa.
- **Search:** The search option gives the user a sophisticated way to browse the attacks and find specific ones. The user can search by any criteria (e.g. source IP addresses, types of attacks, priority, etc).
- **Display header:** This option enables the user to display the header of a selected packet.
- **Send mail:** This option uses sockets and the SMTP to allow the user to send mail. This option allows the user to alert other users about specific attacks. The mail server (imgw.cpsc.ucalgary.ca) was used for testing purposes and can be changed easily to suit the need of the user's network.
- **Add attack to mail:** This option allows the user to select an alert and attach it to a message.
- **Save a record:** This option allows the user to save the list of alerts to a database.
- **Load a record:** This option allows the user to load an old saved record from the database to the Graphical User Interface.
- **Delete a record:** This option allows the user to delete an old saved record from the database.

Using Snort and IDPP together provided an effective way to process largs trace files and made one capable of finding interesting malicious attacks. There are a few screen shots in the next page of IDPP and what follows is a detailed description about the results and some analysis.

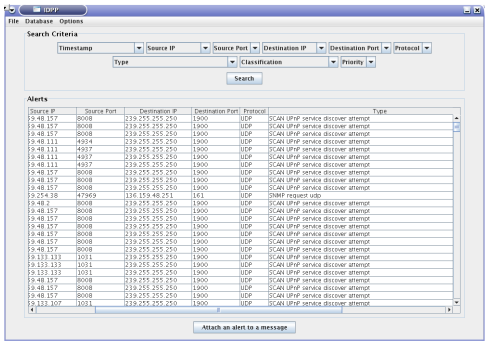

Figure 1: Search

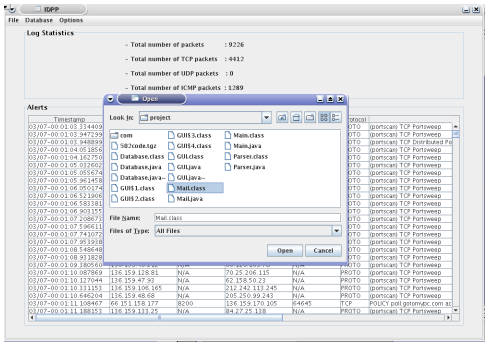

Figure 2: Initial load

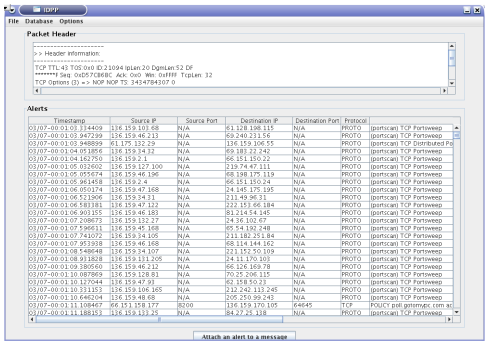

Figure 3: Headers display

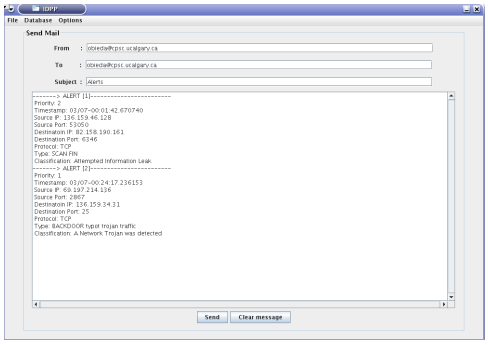

Figure 4: Send mail

5 Results

Trace file captured at the University of Calgary on March 07 2005

Trace		**# of Packets**	**Alerts**	
March 07-2005-21-01-01		6,135,347	7,048	
Attack	**Total**	**From U of C**	**To U of C**	**Other**
BACKDOOR typot Trojan traffic	166	0	166	0
SCAN FIN	822	657	165	0
MISC source port 53 to < 1024	2	0	2	0
TCP Distributed Portscan	78	0	78	0
TCP Decoy Portscan	89	0	89	0
TCP Portscan	383	149	232	2
TCP Portsweep	4,550	4,400	128	22
POLICY poll.gotomypc.com access	951	0	951	0
BAD-TRAFFIC tcp port 0 traffic	7	3	4	0
Total	**7,048**	**5,209**	**1,815**	**24**

Trace file captured at the University of Calgary on March 03 2005

Trace		**# of Packets**	**Alerts**	
March 03-2005-07-01-00		2,997,633	3,410	
Attack	**Total**	**From U of C**	**To U of C**	**Other**
BACKDOOR typot Trojan traffic	162	0	162	0
SCAN FIN	69	2	67	0
TCP Distributed Portscan	10	0	10	0
TCP Decoy Portscan	104	0	104	0
TCP Portscan	317	55	261	1
TCP Portsweep	1,761	1,688	22	51
POLICY poll.gotomypc.com access	954	0	954	0
BAD-TRAFFIC tcp port 0 traffic	33	0	33	0
Total	**3,410**	**1,745**	**1,613**	**52**

Trace file captured at the University of Calgary on March 03 2005

Trace		**# of Packets**	**Alerts**	
March 03 2005-10-01-00		5,956,831	5,646	
Attack	**Total**	**From U of C**	**To U of C**	**Other**
BACKDOOR typot trojan traffic	153	0	153	0
SCAN FIN	1181	1052	129	0
TCP Distributed Portscan	32	0	32	0
TCP Decoy Portscan	104	0	104	0
TCP Portscan	485	147	337	1
TCP Portsweep	2,736	2,661	43	32
POLICY poll.gotomypc.com access	948	0	948	0
BAD-TRAFFIC tcp port 0 traffic	1	0	1	0
SNMP request tcp	3	0	3	0
SNMP trap tcp	3	0	3	0
Total	**5,646**	**3,860**	**1,753**	**33**

Trace file captured at the University of Calgary on March 06 2005

Trace		**# of Packets**	**Alerts**	
March 06 2005-01-01-00		5,115,692	4415	
Attack	**Total**	**From U of C**	**To U of C**	**Other**
BACKDOOR typot trojan traffic	81	0	81	0
DDoS mstream client to handler	1	1	0	0
SCAN FIN	201	104	97	0
TCP Distributed Portscan	36	0	36	0
TCP Decoy Portscan	155	0	155	0
TCP Portscan	238	94	144	0
SNMP trap tcp	5	0	5	0
SNMP request tcp	3	0	3	0
POLICY poll.gotomypc.com access	953	0	953	0
BAD-TRAFFIC tcp port 0 traffic	11	0	11	0
TCP Portsweep	2,731	2,643	49	39
Total	**4,415**	**2,842**	**1,534**	**39**

Trace file captured at the University of Calgary on March 03 2005

Trace		**# of Packets**	**Alerts**	
March 03 2005-22-01-01		5,269,892	6,156	
Attack	**Total**	**From U of C**	**To U of C**	**Other**
BACKDOOR typot trojan traffic	229	0	229	0
DDoS mstream client to handler	18	18	0	0
SCAN FIN	1,013	817	196	0
TCP Distributed Portscan	36	0	36	0
TCP Decoy Portscan	59	0	59	0
TCP Portscan	293	140	147	6
TCP Portsweep	3,534	3,431	86	17
SNMP AgentX/tcp request	4	0	4	0
SNMP request tcp	1	0	1	0
POLICY poll.gotomypc.com access	950	0	950	0
BAD-TRAFFIC tcp port 0 traffic	19	1	18	0
Total	**6,156**	**4,407**	**1,726**	**23**

Trace file captured at the University of Calgary on March 07 2005

Trace		**# of Packets**	**Alerts**	
March 07 2005-06-01-00		3,256,779	4150	
Attack	**Total**	**From U of C**	**To U of C**	**Other**
BACKDOOR typot trojan traffic	86	0	86	0
DDoS mstream client to handler	6	0	6	0
SCAN FIN	697	582	115	0
TCP Distributed Portscan	33	0	33	0
TCP Decoy Portscan	69	0	69	0
TCP Portscan	173	89	69	15
TCP Portsweep	2,132	2,040	33	59
POLICY poll.gotomypc.com access	951	0	951	0
BAD-TRAFFIC tcp port 0 traffic	3	0	3	0
Total	**4,150**	**2,711**	**1,365**	**74**

Trace file captured at the University of Calgary on January 11 2005

Trace		**# of Packets**	**Alerts**	
January 11 2005		1,000,000	6,327	
Attack	**Total**	**From U of C**	**To U of C**	**Other**
SHELLCODE x86 setuid 0	1	0	1	0
BACKDOOR typot trojan traffic	153	0	152	1
DDoS mstream client to handler	2	2	0	0
SCAN FIN	1662	1507	155	0
TCP Distributed Portscan	91	0	91	0
TCP Decoy Portscan	129	0	129	0
TCP Portscan	265	169	96	0
UDP Portsweep	8	8	0	0
TCP Portsweep	3032	2970	21	41
ICMP PING	4	0	4	0
ICMP Large ICMP Packet	1	0	1	0
ICMP PING NMAP	1	0	1	0
POLICY poll.gotomypc.com access	949	0	949	0
BAD-TRAFFIC tcp port 0 traffic	29	11	18	0
Total	**6,327**	**4,667**	**1,618**	**42**

6 Analysis

6.1 Buffer overflow

6.1.1 x86 setuid 0

This is a serious attack which might give the attacker the ability to compromise the target host or open a backdoor for future access. Buffer overflow was described in detail in previous sections. In this senario, the attacker's payload was a setuid code. A

setuid program is one that, if executed, will operate with the permissions of the owner of the program, not of the person executing the program. The target host must have been running a network service whose owner is root and which had a buffer overflow bug. If the attacker delivered the crafted shellcode carefully then the attacker must have got the malicious code to run with root privileges. In other words, the attacker was able to do whatever a root user can do on the target host.

6.2 Trojan horses and backdoors

6.2.1 Typot trojan traffic

Typot is a trojan horse that uses a stealth scanning technique to scan and receive network mapping data. The trojan uses the libpcap and libnet libraries to spoof and capture network traffic. It was designed to infect systems running Linux. Typot begins network mapping by sending a TCP SYN packet every second with a fixed TCP window size of 55808, spoofed source IP and MAC addresses, and a random destination IP address. Typot causes each infected system to enter promiscuous mode and watch for TCP SYN packets that have a window size of 55808. If such packets are captured, then the trojan creates a filename called "r" in the current directory. After 24 hours, the trojan checks whether the file "r" has been created or not. If it was created then the trojan connects to IP address (hardcoded inside the trojan code) via ssh. Once the connection succeeds then Typot simply sends the network mapping data to the hardcoded IP address.

From the results section, Typot's traffic was detected and it seems that a number of hosts on the University's network are being hit by the trojan's traffic.

6.3 Policy

6.3.1 poll.gotomypc.com

The website *www.gotomypc.com* offers a service to access and control any machine remotely from any Internet connection. The service uses the Hyper Text Transfer Protocol (HTTP) and has the potential to open huge security holes in any network. The main goal as claimed by the GoToMyPC company is to offer the user a fast and an easy way to access his machine remotely. When a user subscribes, a password is given to the user to secure the access to user's own machine from any Internet connection. The main problem is that the passwords are stored on the company's servers and if the servers are compromised then one can imagine how easy it would be to access the users machines remotely. Furthermore, if one wants to access his own machine remotely then it is totally insecure to involve a second party. From the results section, all the gotomypc traffic was coming to the University's network. Either some of the University's network users are accessing their own machine on the University's network via the gotomypc service and from other networks or malicious users are using this service to access machines on the University's network. The second possibility is extremely dangerous and can open up a lot of security holes. The best solution is to configure the University's firewalls (the ones that separate the University's network

from the Internet zone) so that it blocks all traffic coming from the *www.gotomypc.com* servers.

6.4 Bad traffic

6.4.1 tcp port 0 traffic

The source ports were set to 0 (an invalid port number) and the destination ports were random ports > 1024. Hence one can conclude that some crafted packets were constructed for scanning purposes. Filtering packets that have their source ports set to 0 can fix the problem.

6.4.2 source port 53 to < 1024

The source ports were set to 53 (DNS) and the destination ports were random ports < 1024. Ports < 1024 are reserved ports and normally no packet will have a source port that is less than 1024. Again one can conclude that crafted packets were constructed for scanning purposes. Filtering any packet with source port < 1024 can fix the problem.

6.5 Denial of Service and Distributed Denial of Service

6.5.1 Large ICMP Packet

This is a Ping of Death attack which was described in detail in previous sections. The attacker attempted to cause the target host to crash by sending it a huge crafted ICMP packet (> 65535 bytes). Many operating systems these days have added patches to do more checking and can drop these huge packets automatically when detected. Hence the best solution is to keep operating systems up to date by installing the latest patches.

6.5.2 mstream client to handler

This is another serious attack which can bring networks down by flooding target machines with large amounts of traffic. Mstream is a Distributed Denial of Service tool which poses the same dangers as earlier DDoS tools (e.g. Tribe Flood, Trin00, etc). Mstream source code consists of 3 files: *master.c* (the handler's code), *server.c* (the zombie's code)and *stream.c* (the payload). The *server.c* once compiled and executed will cause the machine to become a zombie. The attacker can then compile and execute the *master.c* file and control the zombies using telnet. The attacker can direct the zombies to execute the payload in stream file and send TCP ACK packets to the victim hosts using random ports.

From the results section, one can see that all the traffic of mstream is coming from the University's network and which implies that some of the hosts on the network are infected, that is to say, some of the University's hosts executed the server file and became zombies. Mstream's attack slows a machine down by using up CPU cycles. The attack also consumes network bandwidth. In addition to the incoming ACK packets, the target host will consume bandwidth when it tries to send TCP RST packets

to non-existent IP addresses (refer to IP spoofing and DDoS attacks in previous sections for more elaboration). Routers will then return ICMP host/network unreachable packets to the victim, resulting in more bandwidth usage. The distributed method of attack multiplies the effect on the CPU, as well as consuming large amounts of network bandwidth [7].

6.6 Ping and Port Scans

Both Ping and Port Scanning were described in detail in previous sections. In general, network probing tools are used by system and network administrators. They are very useful in troubleshooting networks and enhancing the security countermeasures.

6.6.1 ICMP PING NMAP and ICMP PING

One can observe that the softwares used to send these packets are *nmap* and *ping*. By default, nmap tries to ping the host before scanning it. This can be disabled by using the *-P0* flag. Furthermore, *nmap* can be used to ping a host without using it by using the *-sP* flag. If there are sensitive information on some of the machines on your network then the best solution to prevent such probes is to use a firewall and block all incoming and outgoing ICMP traffic.

6.6.2 SCAN FIN and other port scans

The FIN Scan is one of the stealthy scans that was described in previous sections. Nmap might have been used to send the probes. The best solution to prevent such probes is to use a firewall and block incoming FIN packets to ports that are not listening.

7 Future work and improvements

Most current approaches to detect intrusions utilize some form of rule-based analysis to identify indications of known attacks (e.g. Snort, Bro). Rule-based analysis relies on sets of predefined rules that are provided by an administrator, automatically created by the system, or both. Rule-based detection, however, fail to identify attacks which vary from expected patterns. A minor variations in an attack sequence can affect the activity-rule comparison to a degree great enough to prevent detection. Furthermore, rule-based detection cannot effectively detect attack scenarios that may occur over an extended period of time. In the last few years, an increasing amount of research was done to improve such techniques. A discussion of improvements are described below:

7.1 Honeypots

A honeypot is a system that is designed to make attackers think that its a real system to break into [9]. Examples can be: setting a machine whose only purpose is to log all access attempts, or a machine with unpatched operating system and softwares and which can be exploited. Honeypots have both advantages and disadvantages. Intrusion

detection systems sometimes have problems distinguishing hostile traffic from benign traffic. However, isolated honeypots have a much easier time because these are systems that should not normally be accessed which implies that all traffic to a honeypot system is already suspect. One of the main disadvantages of using honeypots is that if the honeypot is compromised then it might be used to further compromise the network. Furthermore, honeypots add complexity and in security, complexity is bad because it leads to increased exposure to exploits.

7.2 Neural Networks

An artificial neural network consists of a collection of processing elements that are highly interconnected and transform a set of inputs to a set of desired outputs. The result of transformation is determined by the characteristic of the elements and the weights associated with the interconnections among them [3]. By modifying the connections between the nodes the network is able to adapt to the desired outputs. The use of artificial neural networks provides the potential to identify and classify network activity based on limited, incomplete, and nonlinear data sources which can reasolve a number of the problems encountered by the other current approaches to intrusion detection.

The main disadvantage of using such approach is that the ability of the artificial neural network to identify intrusions is completely dependent on the accurate training of the system, the training data and the training methods that are used [2]. The training routine requires a very large amount of data to ensure that the results are statistically accurate. Obtaining a large quantity of data is always a difficult task. Furthermore, the connections weights and transfer functions of various network nodes are usually frozen after the network has achieved an acceptable level of success in the identifications of events. While the network analysis is achieving a sufficient probability of success, the basis for this level of accuracy is not often known due to the black-box nature of the neural network.

8 Conclusion

This paper described in detail the most common attacks used these days to paralyze network resources, and provided traces and strategies for better countermeasure techniques. Data sets captured at the University of Calgary were analyzed using an open source Intrusion Detection System (Snort), and an Intrusion Detection Post-Processor system (IDPP) that was implemented to make the detection and analysis of attacks more effective.

The analysis of the data sets clearly showed how vulnerable a network can be. The collected results showed a number of malicious packets traversing the University's network freely. One packet carrying a buffer overflow shellcode (exploit string) is enough to give any attacker root privileges on the target host. Once the attacker gains root privileges then compromising other hosts is only a matter of time.

Using a well configured Intrusion Detection System provides certain peace of mind. However, it cannot be a replacement for a well managed firewall, regular security

audit, and a strong security policy. The work provided in this paper offers serveral directions for future work and provides a starting point for further investigation into the burgeoning field of Computer and Network Security.

References

[1] Aitel, D., and Young, S. *The Hacker's Handbook*. 2004.

[2] Cannady, J. Artificial neural networks for misuse detection. In *Proc. of the 1998 National Information Systems Security* (1998), pp. 443 – 456.

[3] Cannady, J., Mahaffey, J., and Rhodes, B. Multiple self-organizing maps for intrusion detection. In *Proc. of the 23rd National Information Systems Security* (2000).

[4] Denault, M., Critzalis, D., Karagiannis, D., and Spirakis, P. Intrusion detection: Approach and performance issues of the securenet system. In *Proc. of the 13th National Computer Security Conference* (1994).

[5] E. Kirda, C. K., and Toth, T. Computer security: Service specific anomaly detection for intrusion detection. In *Proc. of the 2002 ACM symposium on Applied computing* (2002).

[6] Fydor. The art of port scanning.

[7] Hatch, B., Kurtz, G., and Lee, J. *Hacking Linux Exposed*. 2003.

[8] Kruegel, C., and Vigna, G. Intrusion detection: Anomaly detection of web-based attacks. In *Proc. of the 10th ACM conference on Computer and Communication Security* (2003).

[9] Northcutt, S., and Novak, J. *Network Intrusion Detection*. 2002.

[10] One, A. Smashing the stack for fun and profit.

[11] Stallings, W. *Cryptography and Network Security*. 2003.

[12] Stanifor, S., Paxson, V., and Weaver, N. How to 0wn the internet in your spare time. In *Proc. of the 11th USENIX Security Symposium* (2002).

[13] Stevens, R. *TCP/IP Illustrated, Volumen I*. 1994.