

Study of Denial of Service

Final Report

Author: Matthew Hutchinson
m.hutchinson@ijug.org

Supervisor: Qiang Gu
Moderator: Prof. A Marshall

Status: Final Report / Dissertation
Date: 27-Aug-03



The Queen's University of Belfast

BRINGING CIVILIZATION TO ITS KNEES...

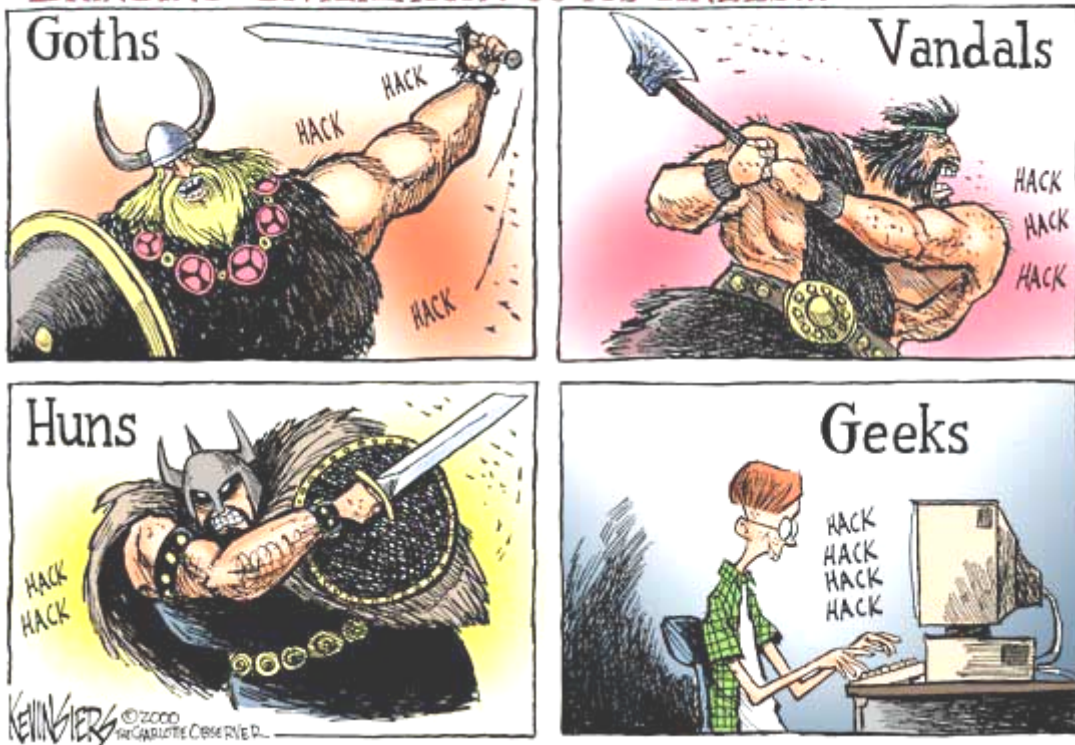


Figure 1 - Denial of service attacks - Kevin Siers, North Carolina - Editorial cartoons from the Charlotte Observer

Abstract

This report details the denial of service (DoS) study. The study begins by defining the origins of DoS and identifying reasons for its use. Some well known attacks are documented, illustrating the effects of an attack. The report then describes four common types of DoS attack; namely, Bandwidth Consumption, Resource Starvation, Programming Flaws and DNS (routing) attacks.

In order to understand protocol exploits in generic DoS attacks, each protocol and its operation is explained in some detail. Development of a SYN flood attacking program is then discussed, the source code for this attack is included in appendix A. The design of the program is improved and a Random SYN flooder is developed (appendix B). Both attacks were found to be effective and denied service to a website running on a local test network.

In order to combat this attack a SYN Detector application was written (appendix C). The software (running on a potential victim) can detect a SYN flood and provide an early warning, before major disruption occurs. The development and simulation of the SYN flood and SYN Detector program is detailed in Chapter 5 – Generic attacks.

Chapter 6 introduces the concept of a DNS attack, launched at the underlying structure of the Internet. Some example attacks are discussed including, PTR record spoofing and the ‘Birthday’ attack. The use of programming flaws in DoS attacks is investigated. The IP fragmentation mechanism was found to contain a number of vulnerabilities that could easily be exploited to create a DoS condition.

Finally the report discusses Distributed denial of service (DDoS), its concept and some example attacks. Nowadays DDoS is the obvious choice for any attacker. They are more complex, effective and harder to trace than standard DoS attacks.

The report concludes that DoS is a powerful attacking tool that all network security experts and programmers should seriously consider. As e-commerce continues to play a major part in the electronic economy, DoS attacks will have a greater impact on our electronic society. A DoS attack can cause loss of revenue (due to downtime), increased security expenses and lack of customer confidence. It can also be used as a propaganda weapon or for political reasons.

This report, the source code, application and workspace files (zipped) are available online at <http://hutchinson.ijug.org/dos>. To report broken links or any other queries related to this study contact me at m.hutchinson@ijug.org



Acknowledgements

I would like to thank the following people, all of whom have had a hand in shaping this project.

To my supervisor Qiang Gu, for his invaluable assistance and guidance throughout, and for the use of his DVD-RW drive and blank CD's.

To Barak Weichselbaum, founder of Komodia.com, for creating and distributing an open source TCP/IP library for Windows XP. And for his help in providing online support while setting up a development environment.

To Tazmen Technologies, for the free distribution of Packetyzer, a network packet analysing tool for Windows. This tool proved to be invaluable during the testing and demonstration of attacking programs.

To Peter Gordon, for lending me the use of his Dell PC and Ethernet hub, which I used to test and launch DoS, attacks.



Contents

1 INTRODUCTION	1
1.1 What is a Denial of Service attack?	1
1.2 The Hacking Community	1
1.2.1 Hacker Demographics	2
1.2.2 Motivation of DoS attackers	2
1.3 Effects of a DoS attack	4
1.4 Reported attacks	4
2 TYPES OF DOS ATTACKS	5
2.1 Bandwidth Consumption	5
2.2 Resource Starvation	5
2.3 Programming Flaws	6
2.4 Routing and DNS attacks	6
2.5 Generic DoS attacks	7
3 COMMON PROTOCOLS	9
3.1 TCP/IP Reference model	9
3.2 TCP/IP Protocol	10
3.3 UDP Protocol	12
3.4 ICMP Protocol	13
3.5 DNS Protocols	14
4 DEVELOPMENT ENVIRONMENT	15
4.1 Software	15
4.2 Hardware	20
5 GENERIC ATTACKS	21
5.1 SYN Flood	21
5.1.1 Attack concept	21
5.1.2 Code walkthrough	23
5.1.3 Simulation	26
5.1.4 Countermeasures	29
5.1.5 Random SYN flooder	30
5.1.6 Attack analysis	32
5.2 SYN Detector	33
5.2.1 Application Design	33



5.2.2 Code walkthrough	34
5.2.3 Simulation	37
5.3 SMURF	40
5.3.1 Attack concept	40
5.3.2 Example Scenario	42
5.3.3 Countermeasures	42
5.3.4 Attack analysis	43
6 DNS ATTACKS	44
6.1 Background Information	44
6.2 PTR Record Spoofing	47
6.3 The Birthday Attack	47
6.4 DNS Bandwidth Consumption	50
7 DoS and Programming Flaws	52
7.1 Ping of Death	52
7.2 IP Fragmentation	53
7.2.1 Fragmentation example	53
7.2.2 Exploits	54
8 Distributed Denial of Service	56
8.1 Attack concept	56
8.2 DDoS attacks	57
9 Conclusions	60
9.1 Summary	60
9.2 Future Work	61
REFERENCES	62

APPENDIX A - Synflooder program source

APPENDIX B - RandSynflooder program source

APPENDIX C - SynDetector application source



Table of figures

Figure 1 – Bringing civilisation to its knees	<i>preface</i>
Figure 2 – DNS Cache Poisoning	7
Figure 3.1 – TCP/IP Reference model	9
Figure 3.2 – TCP Header Structure	10
Figure 3.3 – IPv4 Header Structure	11
Figure 3.4 – UDP Datagram Header	12
Figure 3.5 – ICMP echo or echo reply message header	13
Figure 3.6 – DNS protocol message format	14
Figure 4.1 – Winsock layered architecture	15
Figure 4.2 – Packetizer screenshots	16
Figure 4.3 – Packet capture layered architecture	18
Figure 4.4 – Example NETSTAT command	19
Figure 4.5 – LAN hardware	20
Figure 5.1 – Basic TCP 3-way handshake	21
Figure 5.2 – SYN Flood attack concept	22
Figure 5.3 – SYN Flood formed packet	26
Figure 5.4 – SYN Flood attack simulation	27
Figure 5.5 – Random SYN Flood attack simulation	31
Figure 5.6 – SynDetector and valid SYN packets	37
Figure 5.7 – SynDetector under a SYN flood attack	38
Figure 5.8 – SynDetector under a Random SYN flood attack	38
Figure 5.9 – ICMP echo request packet	40
Figure 5.10 – SMURF attack with amplification	41
Figure 6.1 – DNS packet analysis	44
Figure 6.2 – Birthday Attack Scenario	48
Figure 6.3 – Remote Traffic Interception	49
Figure 7 – IP Fragmentation example	54
Figure 8 – The DDoS attack	57



1 Introduction

1.1 What is a Denial of Service attack?

A denial of service (DoS) attack is a malicious attempt by one or many users to limit or completely disable the availability of a service. They cost businesses millions of pounds each year and are a serious threat to any system or network. These costs are related to system downtime, lost revenues, and the labour involved in identifying and reacting to such attacks. DoS attacks were theorised years ago, before the mass adoption of current Internet protocols. DoS is still a major problem today and the Internet remains a fragile place.

A large number of known vulnerabilities in network software and protocols exist; meaning DoS can be achieved in a number of ways,

- Sending enough data to consume all available network bandwidth (Bandwidth Consumption)
- Sending data in such a way as to consume a resource needed by the service (Resource Starvation)
- Exercising a software ‘bug’ causing the software running the service to fail (Programming Flaws)
- Malicious use of the Domain Name Service (DNS) and Internet routing protocols

Many DoS attacks exploit inherent weaknesses in core Internet protocols. This makes them practically impossible to prevent, since the protocols are embedded in the underlying network technology and adopted as standards worldwide. Today, even the best countermeasure software can only provide a limiting effect on the severity of an attack. An ideal solution to DoS will require changes in the security and authentication of these protocols.

In order to launch some DoS attacks, the programmer must be able to form ‘raw’ packets. Using raw packets, the header information and data can be manipulated to form any kind of packet sequence. Hence techniques such as ‘IP Spoofing’ and malformed ICMP Ping requests can be used.

This report will investigate the mechanism of DoS attacks and their countermeasures, distributed denial of service attacks will also be investigated. A distributed DoS generally has the same effect as a single attack, with the disruption amplified by many systems acting together. These other systems are often compromised machines remotely controlled by the hacker.

1.2 The Hacking Community

In February 2000 ^[22] the arrival of large scale DoS attacks against the commercial sector, prompted many questions from senior executives. How can DoS happen? Why do people do it? Who are these people?

How can they be stopped? In order to determine the origin of these attacks many companies became acquainted with the term ‘hacker’.

1.2.1 Hacker Demographics

A ‘hacker’ was originally defined as ‘*someone who makes furniture with an axe*’, in a modern sense, the definition is, ^[2]

1. A person who enjoys learning the details of programming systems and how to stretch their capabilities. 2. One who programs enthusiastically. 3. A person capable of appreciating hack value (q.v.). 4. A person who is good at programming quickly. 5. An expert in a particular program . . . 6. A malicious or inquisitive meddler who tries to discover information by poking around

Unfortunately, most hackers are branded with the last definition; the correct term for such a person is ‘cracker’. A cracker is an individual who attempts to gain unauthorised access to a computer system. They will often use any means at their disposal to break into a system and perform malicious operations. Examples include extracting credit card details, passwords and other sensitive information. The term ‘cracker’ was coined in 1985 by hackers in defence against journalistic misuse of the term ‘hacker’. ^[2]

‘Cracking’ does not usually involve a huge leap in programming skill, but rather persistence and the dogged repetition of a handful of fairly well-known tricks, that exploit common weaknesses in the security of target systems. Accordingly, most crackers are only mediocre hackers. They tend to form in small, secretive groups that have little overlap with the huge, open hacker culture; though crackers often like to describe themselves as hackers, most true hackers consider them a separate and lower form.

1.2.2 Motivation of DoS attackers

Experienced hackers and the underground community generally view DoS attacks as ‘unsophisticated’, since it is relatively easy for newcomers to download attacking programs from the Internet. It has also become the tool of choice for frustrated crackers. After failing an attempt to perform a malicious operation, a cracker will often bombard the target with a DoS attack. Listed below are some other (less obvious) reasons for DoS attacks.

- **Sub-cultural status** - Because DoS attack programs are easy to download and execute, many new crackers see this as an easy entry point into a close-knit underground group. A DoS attack offers some tangible evidence to prove to the group they are serious about their new ‘career’ as a cracker. (e.g. disabling a prominent website for a number of hours)

- **To gain access** – It is unlikely that any DoS attack will give direct access to a target machine. But they can be used to crash other machines in a network, disabling them while the unauthorised access attempt is made. The attacker may want to disable a logging or tracking service while they gain entry to the system. Crashing a router or firewall may disable some security services.
- **Political reasons** – With the recent conflict in Iraq, the propaganda war took a step into the 21st century. The Al-Jazeera Iraqi media group experienced a distributed DoS attack on its English and Arabic-language websites ^[3]. The network's websites typically receive traffic in the range of 50 or 60 MB/s. During the attack both sites were hit with traffic in excess of 200 MB/s and up to 300 MB/s. The U.S. based company hosting the sites said it could no longer continue to do so, due to the effect of the attacks on other customer's websites.
- **To divert attention** – For example due to a DoS attack, the system administrator has a runaway process on a server causing problems. The attacker uses this opportunity to divert the attention of the administrator while performing other malicious operations on the target.
- **To force a reset** - A 'Trojan' or malicious program has been installed on the victim without the knowledge of the user or system administrator. The attacker needs a system reboot to activate the program. The DoS attack will force the administrator to reboot the server immediately after the crash.
- **To remove reminisces** – A DoS prompted crash may be used when an attacker wishes to cover their tracks very dramatically. Or cover CPU activity with a random crash, hoping to make the administrator think that their activity on the machine was 'just a fluke'.

At present the most common reasons for DoS can be attributed to, unsuccessful, frustrated crackers and those trying to gain sub-cultural status. However, with the recent publicity of these large scale attacks, I believe that political differences may become a more popular reason in the future. The advantages listed below clearly highlight why DoS is gaining popularity among newcomers in the underground community.

- Illegal DoS applications can be downloaded with a GUI front-end for easy use.
- DoS attacks are difficult to detect and prevent.
- Using IP spoofing (faked source IP address); the attacker runs little chance of being caught.
- Distributed DoS (DDoS) attacks offer newcomers a very powerful attacking tool, with little or no programming knowledge required for its use.
- DoS can be a quick and easy way to bring down a victims ISP, disrupting in part access to all other sites hosted by the same provider.

1.3 Effects of a DoS attack

It is often much easier to disrupt the operation of a network or system than to actually gain access. There are a large number of DoS attacks in existence, all targeting vulnerabilities in different services or systems. In general they attempt to shut down or seriously slow down a service provided by a computer system. The effects can range from going unnoticed to disastrous depending on the scale of the attack and the service being targeted.

With the adoption of the Supervisory Control and Data Acquisition Service (SCADA) network in the USA, the risk of a DoS attack could be catastrophic. The SCADA network is an interconnected ribbon of computer systems used for maintaining the nations infrastructure, including power, water and utilities.

1.4 Reported attacks

DoS first received large scale public attention in February 2000 ^[22]. Major Internet sites including CNN, Yahoo, and Amazon suffered a distributed attack over a period of several days. After several months of investigation, law enforcement officials arrested a 15-year-old Canadian youth who used the alias ‘Mafiaboy’ ^[1] and charged him with perpetrating the attacks. In January 2001, the youth pleaded guilty to 56 criminal counts relating to the incident. CNN and other victims claim the attack caused damages totalling \$1.7 billion.

Since then many other attacks have been reported in the media. However it is very likely many large, well known companies do not report DoS attacks, in an effort to protect their corporate image of a secure business. The recent attack on the Al-Jazeera website ^[3] just a few months ago highlights the fact that DoS is still the tool of choice for many individuals.

2 Types of DoS attacks

Although many tools are available to launch DoS attacks, it is important to identify the types that are most often encountered. Described below are four common methods of DoS attack. In these descriptions (and throughout this report) the following terminology will be used;

- *Attacker* – The individual or group responsible for exploiting a DoS condition.
- *Victim / target* – The individual, server or group of machines where the attack will be directed.
- *Flood* – To saturate the victim's bandwidth entirely, or completely deny the use of a service.
- *Countermeasures* – Any steps taken to prevent a denial of service condition, usually through the use of firewall software.

2.1 Bandwidth Consumption

Attackers consume all the available bandwidth on a remote (or local) network. The victim's network connection is saturated by the large volume of traffic generated by the attacker. There are two ways in which this can be achieved.

Larger Pipes

The attacker has a high speed or much faster network connection than the victim. For example the attacker is using a T1 line (1.544 MB/s) to flood a 128 KB/s network link with traffic. It should be noted that this type of attack is not confined to low speed victims. If an attacker can gain illegal access to a network with 100 MB/s of bandwidth, an attack can be launched against a T1 connected victim.

Amplification

Attackers *amplify* their DoS attack by engaging multiple sites to flood the victim's network. Using this process attackers with a slow 22 KB/s connection can completely saturate a T3 (45 MB/s) connection. The attacker must 'convince' the amplifying systems to send traffic to the victim's network. This is usually done by taking advantage of poor security in core Internet protocols, e.g. the Internet Control Message Protocol. (ICMP)

2.2 Resource Starvation

This type of attack targets system resources on the victim's computer (rather than network resources). In doing this the target system is no longer able to operate normally and provide a service across the network. On entering a system the attacker will abuse their allocated quota of system resources to crash the machine. The target system may crash or be forced to reset due to the file system becoming full, processes hanging or CPU utilisation at 100%. Alternatively, if the attacker has managed to gain

unauthorised access, they may choose to simply disable the running service by executing a 'kill' command.

2.3 Programming Flaws

Operating systems, applications or even embedded software all have the potential to fail while handling exceptional conditions. These conditions usually result when a user sends unintended data to the program. Attackers can abuse this vulnerability to send non-compliant packets of data, in an attempt to create a buffer overflow condition and crash the application. For specific applications that reply on user input, attackers can send large data strings thousands of lines long. A service providing application (e.g. web or ftp) running a service with a known flaw could be exploited, rendering that service unavailable.

Instances of programming flaws are also common in embedded logic chips. The infamous Pentium 'f00f' DoS attack allowed a user-mode process to crash any operating system by executing the invalid instruction at 0xf00fc7c8. ^[4]

2.4 Routing and DNS attacks

A routing attack can be devastating and difficult to detect. It involves using the Routing information protocol (RIP v1) and Border Gateway Protocol (BGP v4) to manipulate routing table entries, denying service to legitimate networks. Both RIP v1 and BGP v4 have little or no provision for authentication and security. Victims will have their traffic routed to the attacker's network or to a *black hole* (a network address that does not exist).

DNS attacks involve compromising a Domain Name Server (DNS) and convincing it to cache bad, or incorrect address information. The DNS protocol is inherently vulnerable to this style of attack, due to the weakness of its 16-bit transaction IDs, used during communication with remote systems. When the DNS performs a lookup, it will return the wrong IP address for the domain name, redirecting it to a *black hole*, or the attacker's site. The attacker's site can then pose as the victim's site.

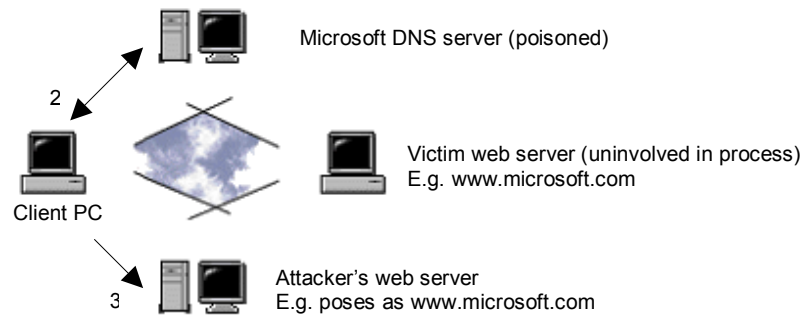
The end-user has no knowledge that they are connected to the wrong website. Since the victim is completely uninvolved in the attack (i.e. no communication takes place between attacker and victim) they have no knowledge traffic intended for their site is being sent elsewhere. This is known as DNS cache poisoning (see figure. 2)

In 1993, Christoph Schuba released a paper entitled "Addressing Weaknesses in the Domain Name System Protocol". ^[5] It outlined several vulnerabilities, including the technique of DNS cache poisoning. In the earliest incarnation, it was possible to provide extra information in a DNS reply packet that would

be cached by the daemon. This allowed an attacker to inject false information into the DNS cache for a network, effectively performing a man-in-the-middle attack.

DNS cache poisoning is now rare. There are new attacks, which make cache poisoning trivial to execute against the large number of name servers running today. The ‘Birthday’ attack is one example that will be discussed later.

Figure 2 – DNS cache poisoning



- 1) Client PC requests www.microsoft.com; the browser tries to resolve IP from Microsoft DNS.
- 2) Cache is poisoned and returns IP for hacker's website.
- 3) User is directed to hacker's website – posing as an exact copy of www.microsoft.com
- 4) Hacker runs background scripts in website to send sensitive information to an inconspicuous email address.
- 5) Hacker checks email account and extracts information as needed. (usually deleting the account after use)

2.5 Generic DoS Attacks

Any DoS attack that does not fall into the categories above can be described as generic. These attacks are capable of affecting many different types of systems. The effects they exhibit on a target machine are similar to the resource starvation and bandwidth consumption attacks. Most of these generic attacks involve some form of protocol manipulation. Internet protocols used in the transport and network layers are usually chosen (e.g. TCP, UDP, IP, ICMP etc.)

IP spoofing is common in most of these attacks. This technique involves the manipulation of the source IP address in the IP header of a transmitted packet. In order to prolong the effectiveness of the attack, attackers use a fake source IP addresses to make tracing and stopping the DoS as difficult as possible. This gives the attacker a form of anonymity. Since DoS is only concerned with consuming bandwidth and resources, in most cases there is no need to worry about properly completing handshakes and transactions.

IP Spoofing is a problem without an easy solution, since it's inherent to the design of the TCP/IP suite.

A distributed denial of service (DDoS) attack can be applied to any type of non-distributed attack, generic or otherwise. Simply by increasing the number of times the attack is simultaneously launched. The effects of the attack are amplified, and even more difficult to trace due to the vast number of sources involved.

Since there are literally hundreds of DoS conditions, with new vulnerabilities being exploited daily, it would be impossible to completely document them all. This report will examine some common DoS mechanisms in detail and provide an illustrated example of each attack. A SYN flood program will be developed with a simple countermeasure application.

3 Common Protocols

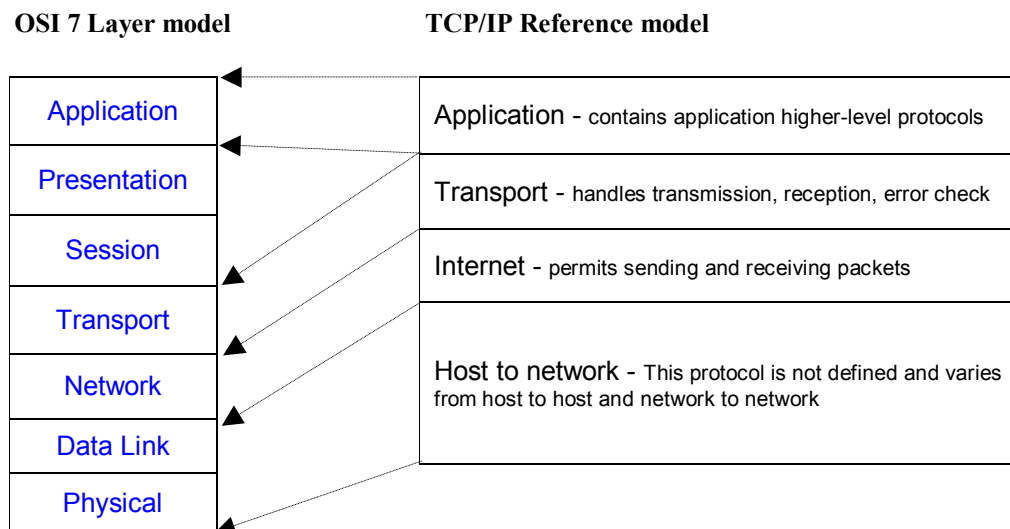
3.1 TCP/IP Reference model

A protocol provides a common set of rules which networked computers use to communicate with each other. It is the manipulation of these rules that make generic DoS attacks possible. Therefore it is necessary to have a basic understanding of how these protocols work under normal operating conditions.

Most network architectures comprise of layers and protocols. Communication from layer n on one host, to layer n on another is conducted according to the protocol defined for that particular layer. Actual communication occurs by passing data down to the layer immediately below, which then adds extra control information before passing it down further. This is repeated until a physical medium is reached. At this point actual communication occurs between the source and destination hosts. The interface between a pair of layers defines which operations and services are offered to its upper layer.

The particular collection of layers and protocols that describe the operation of the Internet is known as the TCP/IP reference model. Figure 3.1 shows the model at its highest level, mapped to the general 7 layer, Open Systems Interconnected (OSI) reference model.

Figure 3.1 – TCP/IP Reference model



Because no need for them was perceived, *Presentation* and *Session* layers are not included in the TCP/IP model. Conceptually, it is useful to envision TCP/IP as a stack, each layer corresponding to a different facet of communication. It should be noted that the TCP/IP reference model can have different protocols in its transport layer, namely UDP or TCP. Some common protocols used in generic DoS attacks will now be discussed.

3.2 TCP/IP Protocol

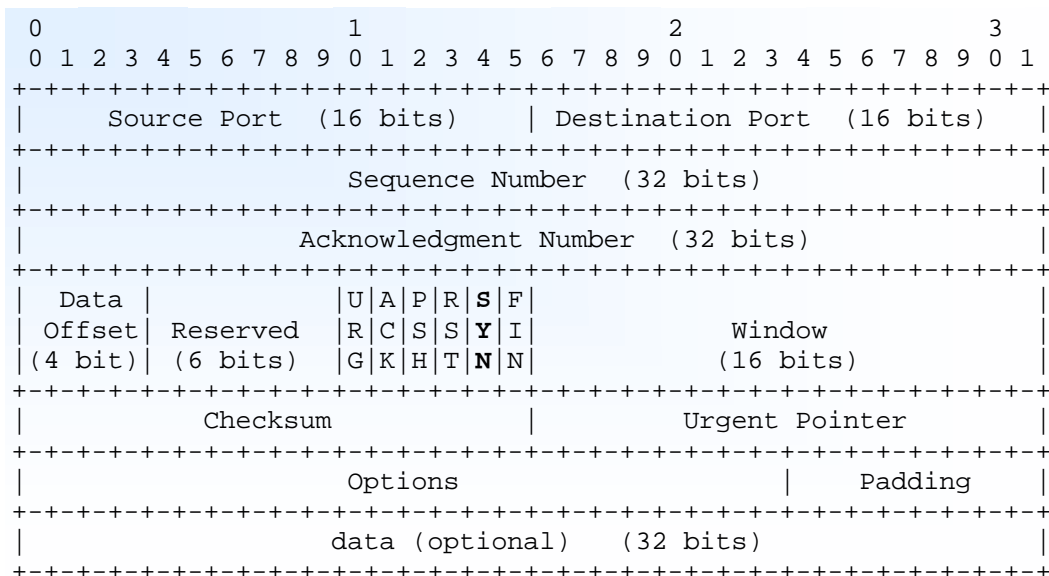
The goal of TCP (Transmission Control Protocol) is to provide a reliable end-to-end byte stream over an unreliable network. It is a transport layer protocol defined in the international document RFC 1122 ^[6], with later revisions in RFC 1323 ^[6].

TCP accepts a stream of data from an application, breaks it up into smaller pieces (not exceeding 64 KB), and passes each *packet* onto the IP layer, to be sent as an IP datagram. When IP datagrams arrive at the destination, they are passed to the TCP process, reconstructed to form the original byte stream, and then sent to the receiving application.

TCP sockets (an IP address and port number) are used to create endpoints between the sender and receiver (the socket connection is full-duplex). TCP is considered as a connection orientated protocol. Meaning a TCP connection must be setup, maintained and released. Whenever an application sends data to a TCP process, it may not be sent immediately. Instead the information may be buffered, in order to collect enough data to ‘fill up’ a packet. Applications can specify that data is immediately transmitted using the **PUSH** flag in the TCP header. Thus TCP operates on the principle of delayed transmission.

The TCP sender also uses a timer. The timer is set to allow the receiving TCP process enough time to send back an acknowledgement packet after receiving the data packet. The acknowledgement packet contains the id of the next segment expected. If the timer runs out and no acknowledgement packet has been received, the packet is sent again. Shown below is the TCP 20 byte header structure (the data + header must fit within the 65,535 byte max. size supported by the IP packet format)

Figure 3.2 – TCP Header Structure



Note - One tick mark represents one bit position.

Sequence Number - The sequence number of the first data octet in this segment (except when SYN is present). If SYN is present the sequence number is the initial sequence number (ISN) and the first data octet is ISN+1.

Acknowledgment Number - If the ACK control bit is set this field contains the value of the next sequence number the sender of the segment is expecting to receive. Once a connection is established this is always sent.

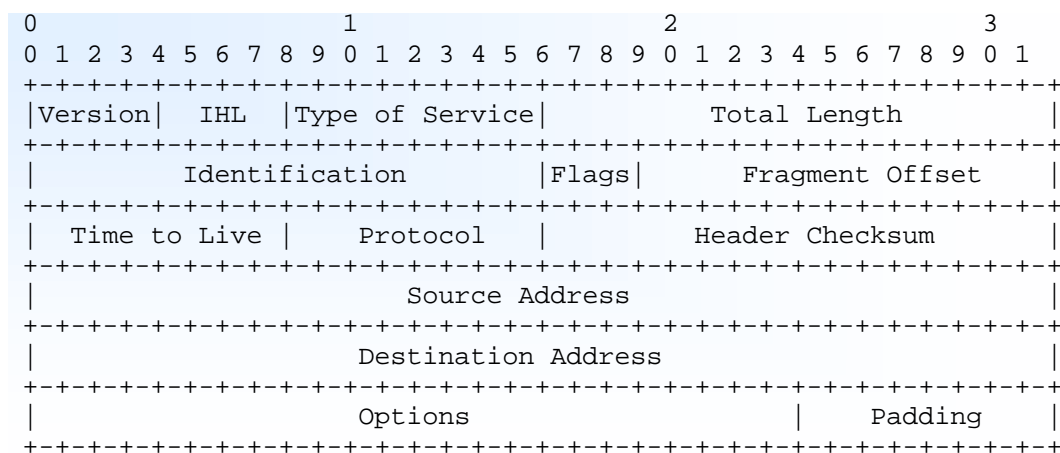
A full description of all header fields and their operation can be found in RFC 1122 ^[6].

Control Bits – (6 bits from left to right URG | ACK | PSH | RST | SYN | FIN e.g. A SYN PACKET 000010)

- URG : Urgent Pointer field significant
- ACK : Acknowledgment field significant
- PSH : Push Function (do not buffer)
- RST : Reset the connection (close sockets)
- SYN : Synchronize sequence numbers (establish connection)
- FIN : No more data from sender (release connection)

The IP protocol is a packet switching protocol, performing addressing and routing between hosts. It is a connectionless network layer protocol, with no acknowledgement. The IP datagram consists of a header and data part. The header structure is a fixed 20 byte part; with a variable length optional part (with a maximum of 40 header bytes). These optional bytes may be used for *record route*, *timestamp route* and other functions.

Figure 3.3 – IPv4 Header Structure



Identification – Used with source address, uniquely identifies the packet for reassembly.

Protocol - Indicates the type of transport packet being carried (6 = TCP; 17= UDP etc.)

Source Address - IP address of the original sender of the packet.

Destination Address - IP address of the final destination of the packet.

A full description of all header fields and their operation can be found in RFC 791 ^[6].

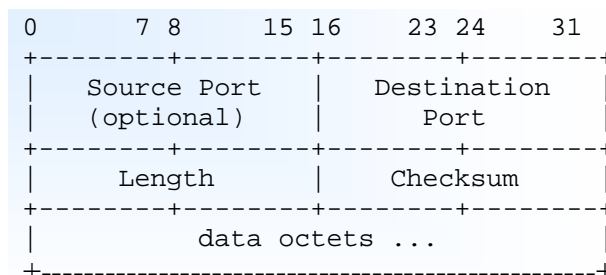
IPv6

Currently, IP version 4 is the standard protocol used in the vast majority of TCP/IP networks. In the early 90's it became apparent that in the future, there would be a shortage of 32 bit IP addresses. IP version 6 emerged as a new standard with 128 bit address fields. IPv6 is a rather large overhaul of all current Internet protocols, extending to Quality of Service (QoS), Security and DNS. It is hoped that when these new protocols are adopted current DoS attacks will be impossible. RFC 2460 ^[6] specifies version 6 of the Internet Protocol (also sometimes referred to as IP Next Generation or IPng.)

3.3 UDP Protocol

UDP is a connectionless transport protocol, entailing that a connection does not need to be established between the source and destination hosts. Like TCP, it uses IP as the underlying network protocol. UDP is usually used when transport speed and performance are paramount. UDP datagrams are passed to the IP layer for routing. A simple UDP datagram can be as large as 64 KB. It consists of a small 8 byte header and the data to be transferred. The UDP protocol is fully defined in RFC 768 ^[6]

Figure 3.4 – UDP Datagram Header



This protocol provides a method for applications to send messages to other programs with minimum overhead data. The protocol is transaction oriented; delivery and duplicate protection are not guaranteed. Messages may arrive out of sequence at the destination host, due to different IP routing paths chosen in transit. The UDP protocol does not provide any mechanism for datagram acknowledgement. Therefore applications requiring ordered, reliable delivery of streams of data should use TCP.

Length - the length in octets of this user datagram including the header and data.

Source Port - An optional field, when meaningful, it indicates the port of the sending process, and may be assumed to be the port to which a reply should be addressed in the absence of any other information. If not used, a value of zero is inserted.

A full description of all header fields and their operation can be found in RFC 768 ^[6].

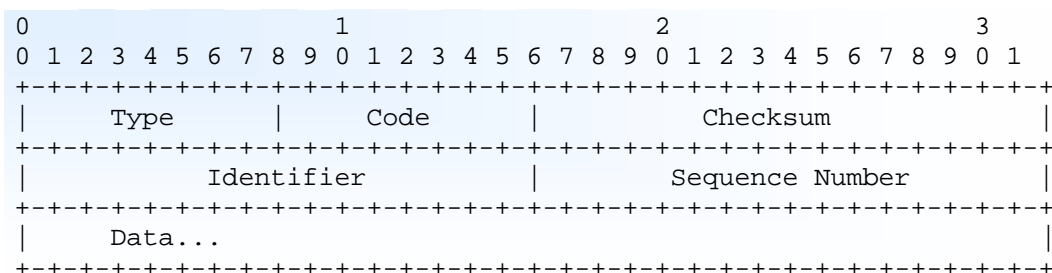
3.4 ICMP Protocol

The Internet Control Message Protocol (ICMP) documented in RFC 792, ^[6] is an error reporting and diagnostic protocol, it is considered a required part of any IP implementation. The protocol is tightly integrated with IP, hence packet delivery is unreliable. A host cannot count on receiving ICMP packets for any network problem. The main functions of the protocol are to:

- **Announce network errors**, such as a host or entire portion of the network being unreachable.
- **Announce network congestion**. When a router begins buffering too many packets, due to an inability to transmit them as fast as they are being received, it will generate ICMP Source ‘Quench’ messages. These should cause the rate of packet transmission to be slowed.
- **Assist Troubleshooting**. ICMP supports an ECHO function, which sends a packet on a round-trip between two hosts. Ping, a common network management tool, is based on this feature. Ping will transmit a series of packets, measuring average round-trip times and computing loss percentages.
- **Announce Timeouts**. If an IP packet's TTL field drops to zero, the router discarding the packet will often generate an ICMP packet announcing this fact.

There are 11 ICMP message types in total, each with a corresponding type code. A typical ICMP packet header is shown below (for an ECHO or ECHO reply message.) RFC 792 ^[6] completely defines all message type headers. The ICMP header format differs depending on the message type being transmitted.

Figure 3.5 – ICMP echo or echo reply message header



The IP address field of the source in an ECHO message will be the destination of the ECHO reply message. To form an ECHO reply message, the source and destination addresses are simply reversed, the type code changed to 0, and the checksum re-computed. Many DoS attacks exploit the use of the ICMP ECHO message (and hence the Ping command.) For example the SMURF and Ping of Death attacks.

3.5 DNS protocol

Domain Name Services (DNS) provide a means of mapping between a numeric IP address and an ASCII string. The term DNS can refer either to the entire system, or to the protocol that makes it work. The actual DNS protocol is used to request resource records from name servers. TCP or UDP can be used to transport DNS protocol messages, connecting to server port 53 for either. However resource records lookups are usually done with UDP. In this case an ‘intelligent retransmission’ scheme must be employed (though one is not specified in the protocol.) The protocol itself is stateless; all the information needed is contained in a single message, fully documented in RFC 1034 ^[6], and having the following format:

Figure 3.6 – DNS protocol message format

Header	
Question	Question for the name server
Answer	Resource Records answering the question
Authority	Resource Records pointing toward an authority
Additional	Resource Records holding additional information

Summary

The operation of these protocols will be examined in greater detail, when relating their vulnerabilities to a specific DoS attack. Often the ideal solution to these protocol based attacks, is a redesign of the protocol mechanism. In all cases this is impractical; it would require a massive overhaul of network software and equipment across the Internet. Therefore alternative approaches must be found to strengthen the weaker aspects of protocol authentication and security. It will become evident that some protocols are inherently more secure than others. Some generic DoS attacks will now be discussed.

4 Development Environment

4.1 Software

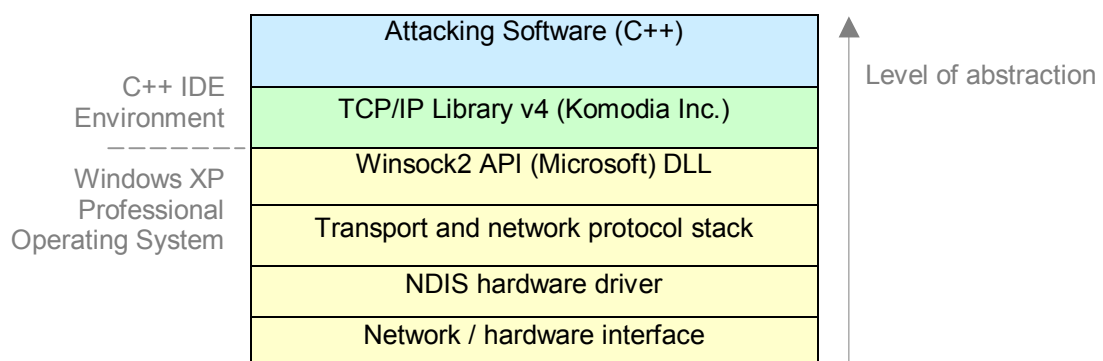
A Windows environment was chosen for developing attacking programs, and for acting as the victim machine. Unfortunately Windows 95, 98 and 2000 do not allow the creation of ‘raw’ sockets. Raw sockets must be available in order to write a generic DoS attacking program. The sockets are described as ‘raw’, because they allow the programmer to completely modify and manipulate every byte of data in the protocol header fields. Thus the programmer has full control over what is sent across the network at byte level.

Windows operating systems use an interface known as Winsock, between the Session and Transport layers. This interface does not provide support for creating raw sockets, and the only alternative is to write a TDI/NDIS driver (data-link layer) – to interface directly with the network card hardware. This would be a long and complicated task, and can be avoided by using Windows XP Professional.

The obvious choice would be to use UNIX or Linux to develop an attacking program, since its kernel provides built in support for raw packet transfers. I chose Windows XP since it is an operating system I have more experience with, and it allows the creation of raw sockets when the user is logged in as an administrator. Microsoft chose to include this feature, much to the dismay of network security experts, but perhaps in response to claims that UNIX has a ‘more powerful’ development environment.

Windows XP uses the Winsock2 API. Rather than write directly to this API, I chose to make use of an open source 3rd party library provided by Komodia Inc ©. TCP/IP library v4 is a C++ library that encapsulates Winsock2 API functions, and allows the user to control every aspect of socket operations ^[7]. The library is free to download from Komodia.com. It provides many built in functions and data types related to raw sockets. Full documentation on each class can be found in reference ^[7]. The diagram below shows the level of abstraction from the network card to the attacking software.

Figure 4.1 – Winsock layered architecture



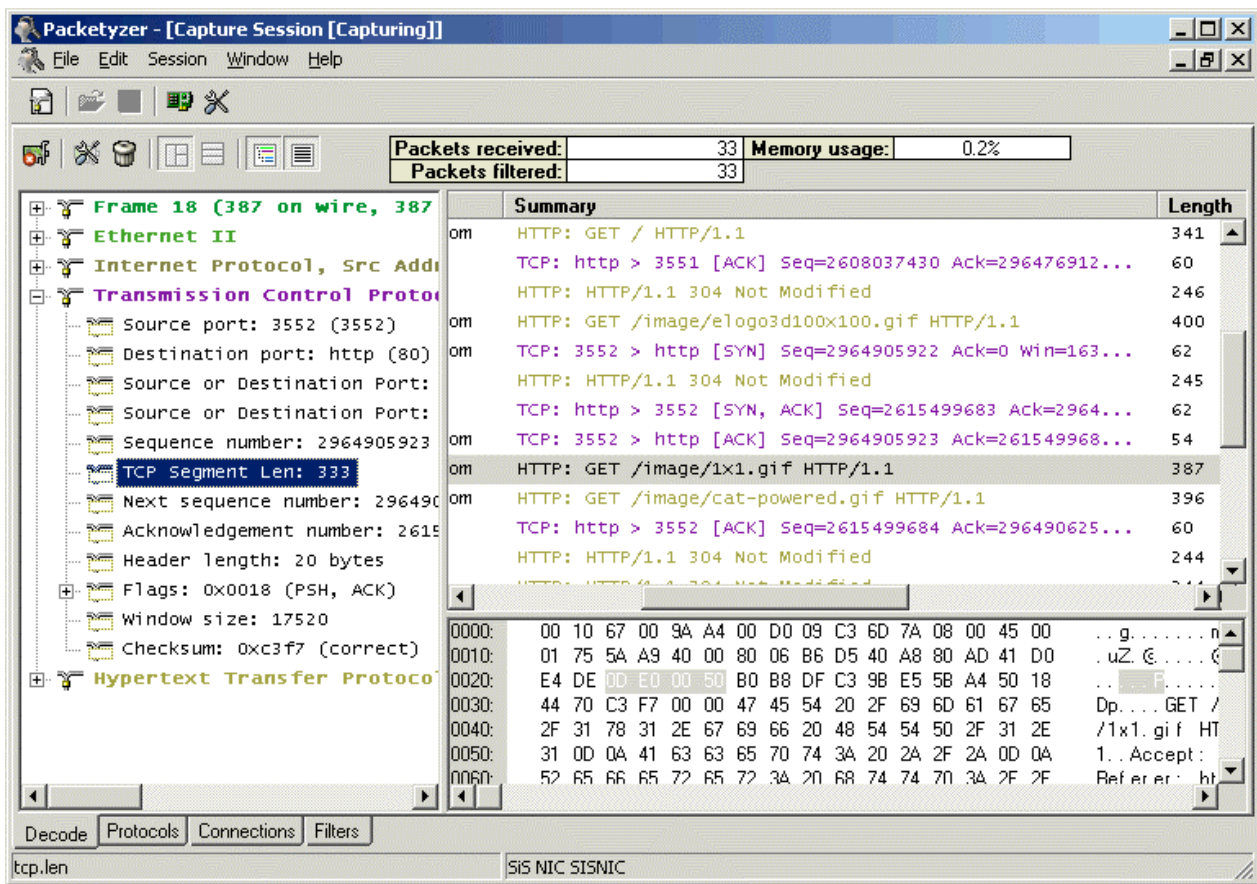
The Microsoft Visual C++ IDE (version 6) was chosen to author, compile and link the attacking programs to the Komodia library. The Windows QoS packet scheduling feature must be disabled for the attacking network card. Since QoS packet scheduling may interfere with the attacking program, as packets are sent across the network. This can be configured in the Windows XP control panel.

The following software is required to analyse, record and monitor network traffic on the Windows platform. The Packetyzer™ application [8] is freely distributed by Tazmen Technologies, and proved to be an invaluable tool for attack analysis. It is particularly useful for verifying protocol based attacks on both the target and victim hosts.

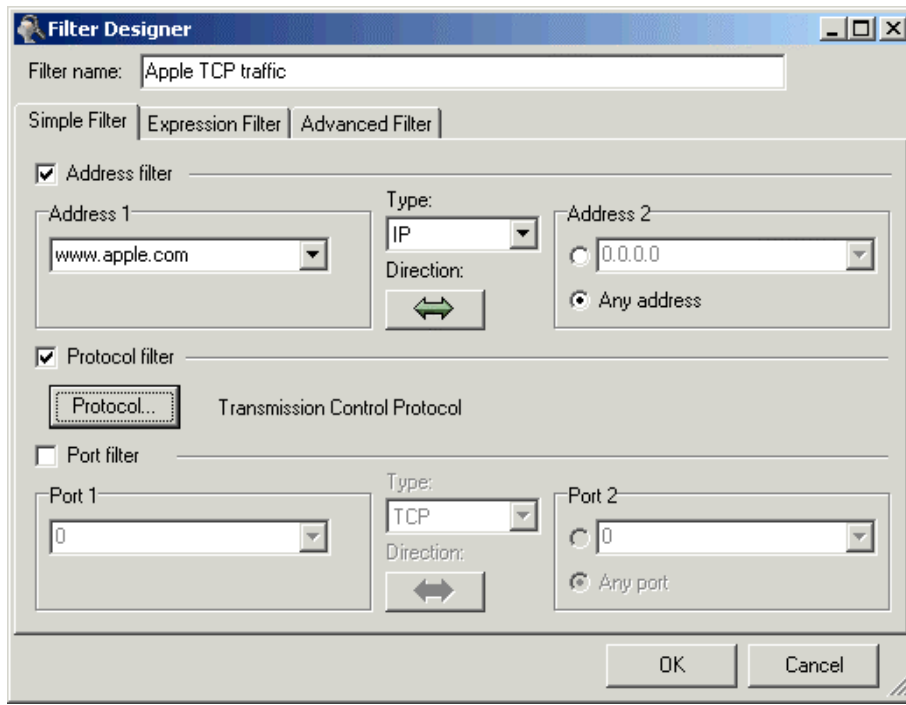
Packetyzer

Packetyzer [8] is a Windows user interface for the Ethereal packet capture and dissection library. It provides an intuitive front-end GUI for packet assembly, analysis and recording. The code is open source and freely distributed under the GNU Public License. The application can be configured with user-defined traffic filters, to display and capture network traffic from any network interface card (NIC). It also has a TCP flow feature which provides a graphical display of TCP packet transfers.

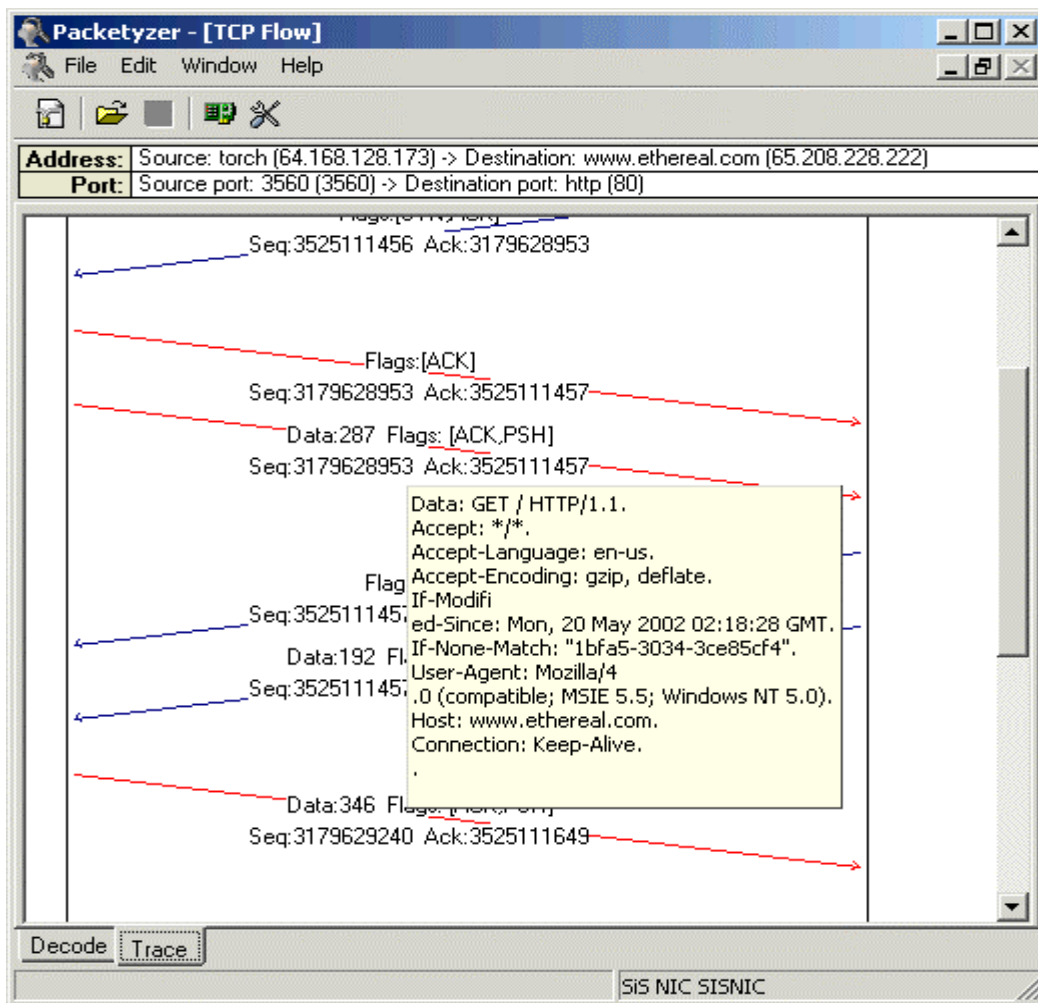
Figure 4.2 – Packetyzer screenshots



Packet decode view. Packet content displayed in left frame, with full byte analysis of packet in lower right frame.



Filter designer showing a simple filter. Incoming / outgoing packets can be filtered by IP, port, protocol or any other header field.



TCP packet flow analysis, shown in graphical view. The SYN, ACK, RST flags are clearly visible with each sequence number.

Packetyzer must have the WinPcap architecture installed. WinPcap v2.3 is distributed with Packetyzer v0.6.6, but since WinPcap v3.0 was available I chose to install it.

WinPcap

WinPcap ^[9] is an architecture for packet capture and network analysis on Win32 platforms. It includes a kernel level packet filter, a low level dynamic link library (packet.dll), and a high-level and system-independent library (wpcap.dll). Packetyzer uses WinPcap, since it needs raw access to the network interface without the intermediation of entities like protocol stacks. WinPcap can provide facilities to:

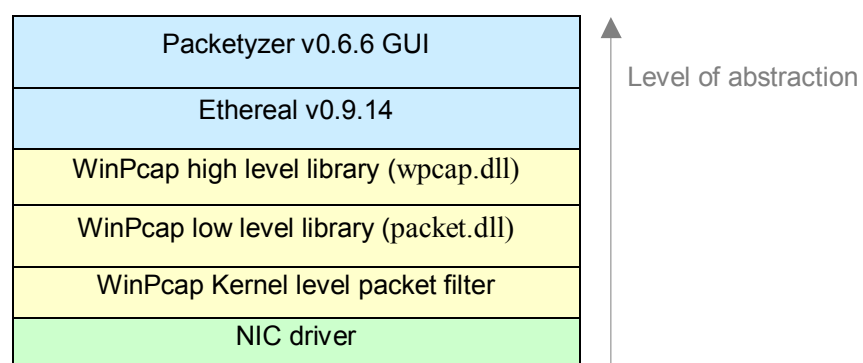
- capture raw packets
- filter the packets according to user-specified rules before dispatching them to the application
- transmit raw packets to the network
- gather statistical values on the network traffic

This set of capabilities is obtained by means of a device driver that is installed inside the networking portion of the Win32 kernels, and through DLLs. All these features are exported through a powerful programming interface, easily exploitable by other applications and portable on different operating systems.

Ethereal

Ethereal ^[10] is a free network protocol analyzer for UNIX and Windows. It can examine data from a live network or from a capture file on disk. You can interactively browse the capture data, viewing summary and detailed information for each packet. Ethereal has several powerful features, including a rich display filter language and the ability to view the reconstructed stream of a TCP session. Packetyzer is basically an enhanced front-end adaptation of Ethereal, with some extra features. Shown below is a diagram of how the packet capture software is layered.

Figure 4.3 – Packet capture layered architecture

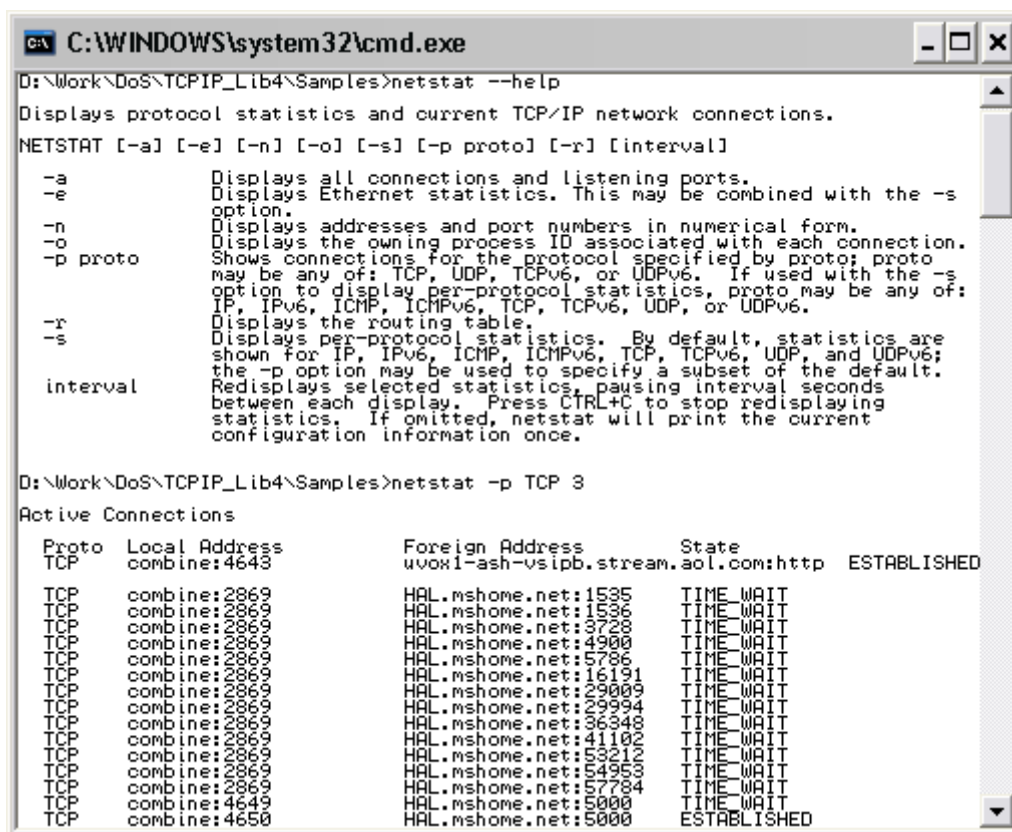


Netstat

NETSTAT is a standard Windows command line tool, used to display protocol statistics and current TCP/IP connections. It can be used to view the status of open or half-open connections. Used without parameters, NETSTAT displays active TCP connections. The command can also be configured to automatically refresh its output every few seconds. The example below illustrates how NETSTAT can be called to show only TCP connections every 3 seconds, on both Internet and LAN interfaces (the LAN connection is between the ‘combine’ and ‘HAL’ computers.)

NETSTAT is actually a collection of diagnostic tools. It can display active TCP connections, ports on which the computer is listening, Ethernet statistics, the IP routing table, IPv4 statistics (for the IP, ICMP, TCP, and UDP protocols), and IPv6 statistics (for the IPv6, ICMPv6, TCP over IPv6, and UDP over IPv6 protocols).

Figure 4.4 – Example NETSTAT command



```

C:\WINDOWS\system32\cmd.exe
D:\Work\DoS\TCPIP_Lib4\Samples>netstat --help
Displays protocol statistics and current TCP/IP network connections.
NETSTAT [-a] [-e] [-n] [-o] [-s] [-p proto] [-r] [interval]
-a          Displays all connections and listening ports.
-e          Displays Ethernet statistics. This may be combined with the -s
           option.
-n          Displays addresses and port numbers in numerical form.
-o          Displays the owning process ID associated with each connection.
-p proto    Shows connections for the protocol specified by proto; proto
           may be any of: TCP, UDP, TCPv6, or UDPv6. If used with the -s
           option to display per-protocol statistics, proto may be any of:
           IP, IPv6, ICMP, ICMPv6, TCP, TCPv6, UDP, or UDPv6.
-r          Displays the routing table.
-s          Displays per-protocol statistics. By default, statistics are
           shown for IP, IPv6, ICMP, ICMPv6, TCP, TCPv6, UDP, and UDPv6;
           the -p option may be used to specify a subset of the default.
           Redisplays selected statistics, pausing interval seconds
           between each display. Press CTRL+C to stop redisplaying
           statistics. If omitted, netstat will print the current
           configuration information once.

D:\Work\DoS\TCPIP_Lib4\Samples>netstat -p TCP 3
Active Connections
Proto Local Address           Foreign Address           State
TCP   combine:4643             uvox1-ash-vsipb.stream.aol.com:http ESTABLISHED
TCP   combine:20669            HAL.mshome.net:1535      TIME_WAIT
TCP   combine:20669            HAL.mshome.net:1536      TIME_WAIT
TCP   combine:20669            HAL.mshome.net:3728      TIME_WAIT
TCP   combine:20669            HAL.mshome.net:4900      TIME_WAIT
TCP   combine:20669            HAL.mshome.net:5786      TIME_WAIT
TCP   combine:20669            HAL.mshome.net:16191     TIME_WAIT
TCP   combine:20669            HAL.mshome.net:39009     TIME_WAIT
TCP   combine:20669            HAL.mshome.net:39994     TIME_WAIT
TCP   combine:20669            HAL.mshome.net:36343     TIME_WAIT
TCP   combine:20669            HAL.mshome.net:41102     TIME_WAIT
TCP   combine:20669            HAL.mshome.net:53212     TIME_WAIT
TCP   combine:20669            HAL.mshome.net:54953     TIME_WAIT
TCP   combine:20669            HAL.mshome.net:57734     TIME_WAIT
TCP   combine:4649             HAL.mshome.net:5000      TIME_WAIT
TCP   combine:4650             HAL.mshome.net:5000      ESTABLISHED

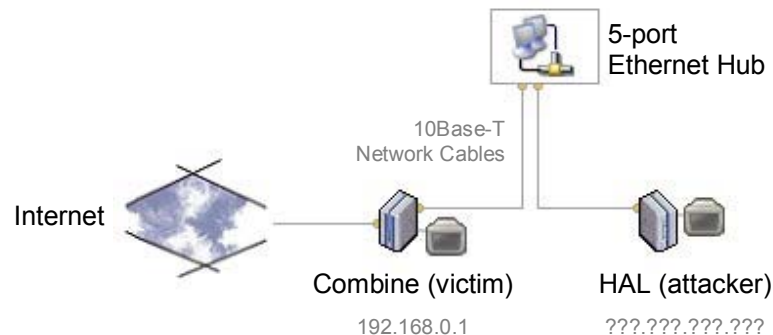
```

With both NETSTAT and Packetizer, it is possible to monitor all network activity between two hosts. This is useful in examining generic protocol based attacks. In other types of DoS attacks (e.g. bandwidth consumption, programming flaws) it can be difficult to show the progress or effect of the attack, from the attacking computer. The actual denying of the service becomes the only way to verify the attack was successful.

4.2 Hardware

In all attacking simulations a small Local Area Network (LAN) will be used. The network comprises of just two computers and a network hub. A complete description of each PC is detailed below. Only one computer (combine) is connected to the Internet. The other uses Microsoft Internet Connection Sharing (ICS) software. This Internet connection must be disabled (on combine) while any simulation is running. Injecting raw packets and performing DoS attacks through a public network would be considered illegal if detected. The computers have the network names ‘combine’ (victim) and ‘HAL’ (attacker) and will be referred to as such throughout this report. The LAN runs at 10 Mbps and is routed through a 5-port Ethernet hub. Using this hub the IP address of the victim machine stays constant (192.168.0.1), while the attackers IP is automatically assigned by Windows during start up.

Figure 4.5 – LAN hardware



<u>Combine (victim)</u>	<u>HAL (attacker)</u>
Windows XP Professional SP1	Windows XP Professional SP1
AMD Athlon 2000XP	AMD Athlon 2000XP
512Mb DDR RAM	512 Mb DDR RAM
512 Kbps Cable Internet connection	Shared internet connection (512 Kbps)
30GB and 20 GB HD	40 GB HD
NIC 1 – Network card for cable internet	NIC 1 – Network card for 10 Mbps LAN
NIC 2 – Network card for 10 Mbps LAN	
Apache HTTP Server 2.0.47	Packetizer v0.6.6
Other vulnerable applications and services	WinPCap v3.0
Packetizer v0.6.6	
WinPCap v3.0	

Both machines have Packetizer installed for packet capturing and monitoring. The victim machine hosts a variety of applications. The Apache web server (running on port 80) is one application at which DoS attacks will be launched.

5 Generic Attacks

5.1 SYN Flood

The SYN Flood attack was one of the first DoS attacks to cause mass disruption among public servers. Panix.com, a large Internet Service Provider (ISP) providing Internet access to several hundred thousand New York inhabitants, came under heavy fire on Sept. 6, 1996 with one of the worst SYN Flood attacks ever seen. Their entire customer base including some large e-commerce firms were out of operation for a number of days. The attack exploits a rather obvious vulnerability in the connection setup stage of the TCP/IP protocol. Unfortunately during the design of TCP/IP the Internet was considered to be a ‘friendly’ place and little consideration was given to the idea of DoS attacks.

5.1.1 Attack concept

When a TCP connection is initiated a three step process (often referred to as a three-way-handshake) occurs. In the figure below, TCP A and B are TCP processes running on different hosts. TCP B begins in the LISTEN state, ready to accept any incoming connection requests. Looking at a normal TCP handshake operation we have;

Figure 5.1 – Basic TCP 3-way handshake

TCP A	TCP B
1. CLOSED	LISTEN
2. SYN-SENT --> <SEQ=100><CTL=SYN>	SYN-RCV
3. ESTABLISHED <-- <SEQ=300><ACK=101><CTL=SYN, ACK>	<-- SYN-RCV
4. ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK>	ESTABLISHED
5. ESTABLISHED --> <SEQ=101><ACK=301><CTL=ACK><DATA>	--> ESTABLISHED (Data sent)

Note -TCP states represent the state AFTER the departure or arrival of a segment
(Whose contents are shown in the centre of each line, CTL = control flags set)

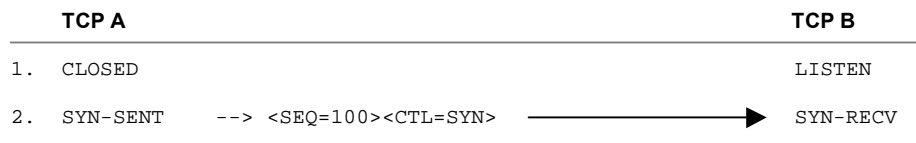
In line 2, TCP A begins by sending a SYN segment indicating that it will use sequence numbers starting from 100. In line 3, TCP B sends a SYN-ACK and acknowledges the SYN it received from TCP A. Note that the acknowledgment field indicates TCP B is now expecting to hear sequence 101, acknowledging the SYN which occupied sequence 100.

At line 4, TCP A responds with an empty segment containing an ACK for TCP B's SYN; and in line 5, TCP A sends some data. The sequence number of the segment in line 5 is the same as in line 4 because the ACK does not occupy sequence number space.

Although this mechanism works for all valid TCP requests, attackers can leverage a weakness in this system to create a DoS condition. The problem occurs due to the fact that most systems allocate a finite number of resources when setting up a ‘potential’ connection or a connection that has not yet been established. Although many systems can sustain hundreds of concurrent connections to the same port (e.g. port 80 for http), it may only take a dozen or so ‘potential’ connection requests to exhaust all resources allocated to setup a new connection. Thus SYN Flooding is a resource starvation attack.

When a SYN Flood is initiated the following sequence occurs;

Figure 5.2 – SYN Flood attack concept



(This packet contains a spoofed source address of a non-existent system – it is sent repeatedly)

The attacker changes the source address of the SYN packet sent to TCP B. TCP B will then try to send a SYN/ACK packet to this spoofed address. Normally if the spoofed address were an actual system, it would respond to TCP B with a RST (reset) packet, since it did not initiate the connection. However the attacker chooses to use the spoofed address of an unreachable system. Thus TCP B never receives a RST packet, and the ‘potential’ or ‘half-open’ connection in the SYN_RECV state is placed in a connection queue. This ‘potential’ connection will only be flushed after a connection establishment timer expires. This timer can vary from 75 seconds to as long as 23 minutes on some systems.

Because the connection queue is small, attackers may only have to send a few spoofed SYN packets every 10 seconds to completely disable a port. The victim system will never be able to clear the backlog of half open connections before receiving a new spoofed SYN packet. When multiple SYN flood packets are directed at a specific port on the victim machine, the service running on this port becomes starved of its resources.

Hence a web server (port 80) or ftp server (port 20/21) can be disabled for the duration of the attack. If the attack is severe enough (i.e. more resources consumed) the service or operating system may crash, causing more disruption. This forces the administrator to reset the computer.

Some variations on the standard SYN flood randomize the unreachable source address. This is necessary when attacking SunOS, Linux or BSD systems since they use cookies in their TCP/IP implementation. This report will be concerned with developing a SYN flood program. The source code and simulation of this program will now be discussed.

5.1.2 Code walkthrough

Appendix A contains the source code for the SYN Flood attacking program. Code comments fully describe its operation. The program uses the Komodia ^[7] 3rd party library (discussed in chapter 4.1 ^{page 15}) to implement raw packet construction and raw socket support. This walkthrough serves to highlight important functions in the code and give an overview on how the attack is launched.

The Komodia library provides a CTCP Crafter object for creating and sending custom TCP/IP packets across a network. The SYN flood program creates a CTCP Crafter object, assigns values to its header fields and sends it repeatedly to a specific port on the victim machine. The packet is sent with a spoofed source IP address, the SYN flag set and no data. The CTCP Crafter object provides methods to validate, create, manipulate and send raw packets. TCP and IP header checksums are also generated by the library with a call to CTCP Crafter.SetUseDefaultChecksum(TRUE).

A FLOODSLEEP parameter is defined to specify a delay (milliseconds) between the sending of each packet. Injecting packets too fast causes many to get dropped as the receiving kernel can't cope. This is especially true on Ethernet connections (10 MB/s) – A call to Sleep(FLOODSLEEP) is introduced after each packet is sent. FLOODSLEEP is initially 100ms, but this value can be altered before compiling. The program also allows the user to cancel the flood while it is in progress by pressing any key. A description of each method now follows.

The attack program is executed from the command line. The argument usage is;

```
Synflooder <unreachable-host> <target-host> {port-num} {num-times}
e.g. Synflooder 100.0.0.1 yahoo.com 80 2000
```

- <unreachable-host> - the source IP address to be used in crafted packets (IP or domain name)
- <target-host> - victim's IP address or destination address for crafted packets (IP or domain name)
- {port-num} - the destination port to attack on the victim (optional - defaults to port 80)
- {num-times} – the number of times to send the SYN packet (optional - defaults to 1000)

GeneralErrorMessage(const char*)

Displays an error message to the screen and closes all open sockets. This method is called before the program exits due to an error or irregular operation.

SetIPCrafter(CIPCrafter)

This method sets up the IP header fields for the crafted packet. The main method passes a CTCP Crafter object, and a CIPCrafter object is obtained from this argument. The method then assigns the IP fields; header length, type of service, and fragmentation flags. It also sets the IP header checksum which is automatically generated by the library.

```
string resolveIPAddress(CTCPCrafter, char*)
```

If the argument passed is a domain name, this method tries to resolve a valid IP address (returned in a string). The address is resolved using the `CTCPCrafter.ResolveDNS()` method. If no valid IP can be found or the host is unreachable the method will display an error message and return an empty string.

The argument passed can also be a numerical IP address, in this case the string length is checked and the `CTCPCrafter.ValidAddress()` method is used to check the address is valid. Again, an empty string is returned in the case of an error. Note: if a numerical IP is passed no DNS lookup is performed, the method only checks for correct syntax and formation of the address. Hence the unreachable host argument (from the command line) should be specified as a numerical IP, not a domain name. Since a DNS lookup on an unreachable domain name will never resolve.

```
int main(int, char*)
```

The main program activity occurs in this method. First a check is made to insure at least the un-reachable host and target address have been specified on the command line. If not, usage information is displayed and the program exits. A call to `CSpoofBase::InitializeSockets()` is made. This is required by the Komodia library before any socket communication can take place.

A new `CTCPCrafter` object is created, `pCrafter`. This is used to resolve and validate each command line argument address (using the `resolveIPAddress()` method.) If the target host is unreachable, or the unreachable source IP address is malformed, the program will exit with an error message. TCP header fields are then defined and assigned to the `CTCPCrafter` object (`pCrafter`). Some header fields to note are:

- Source port number, `sourcePort = 37015`

This is a randomly chosen source port for the TCP header; the port number is incremented before each packet is sent. This port number is irrelevant in a SYN flood, since no communication between ports is ever established. Port numbers range from 0 to 65,535 (16 bit) so an unsigned short variable is used.

- TCP sequence number, `uiSequence = 3908324485`

Under normal TCP/IP operation, the initial sequence number from the source is randomly generated. The destination host then provides its sequence number in the reply. For a SYN flood the only concern is the source TCP sequence number. In this case it has been randomly chosen as 3908324485, a value taken from a typical valid packet. To prevent conflicts with other SYN connection attempts, this number must be incremented before sending each packet. In normal circumstances sequence numbers from the same host to different TCP/IP connections are incremented by 128,000. Since we are spoofing the source IP address we need only increment by one.

- IP identification field, `usIdentification = 0x159d`

The IP identification field is used to distinguish the fragments of one datagram from those of another. The originating protocol module of an Internet datagram sets the identification field to a value that must be unique for that source-destination pair and protocol for the time the datagram will be active in the network. Again an unsigned short (16 bit) variable is used, and a typical valid value (0x159d) is incremented before each SYN packet is sent.

- Packet data , `dataString = “ ”`

The program provides functionality for sending data in the packet. This is not necessary in a SYN flood attack. However attackers have been known to leave messages or their ‘signature’ in this data field (e.g. In the 1996 PANIX.com attack, the attacker included his ‘handle’ and a short message in each SYN packet.) Using the `dataString` variable, character data can easily be added to the packet. The `dataSize` is determined before sending.

After setting TCP header fields and generating the TCP header checksum, IP header fields are assigned with a call to `SetIPCrafter(pCrafter)`. The source IP address and Time to Live (TTL) fields are then set. The time is measured in units of seconds, but since every module that processes a datagram must decrease the TTL by at least one, even if it processes the datagram in less than a second, the TTL can be considered as an upper bound on the time a datagram may exist. The intention is to cause undeliverable datagrams to be discarded. Setting this value to maximum (255) insures the packet reaches its destination without timing out. Typically a value of 128 - 255 is used.

An attempt to create the packet is then made with a call to `pCrafter.Create()`. If any errors occur during packet creation, e.g. malformed header fields, the program will exit. The packet is then sent to the victim by calling `pCrafter.SendRaw(sourcePort, dstIP, portNum, pData, dataSize, 0)`.

This packet header assignment, creation and sending all takes place in a simple for loop. The loop executes a number of times, specified by the command line argument `numTimes` (default is 1000.) The header fields that must change during loop iteration are;

1. (TCP) Source Port number `sourcePort++`
2. (TCP) TCP sequence number `uiSequence++`
3. (IP) IP Identification field `usIdentification++`

Successfully sent packets are indicated to the user by echoing a ‘.’ character back to the command line. The flood can be cancelled at any time by pressing any key during the attack. After the main for loop has finished, all sockets are closed and the program exits. An example of a typical SYN packet sent by the program is shown below.

Figure 5.3 – SYN Flood formed packet

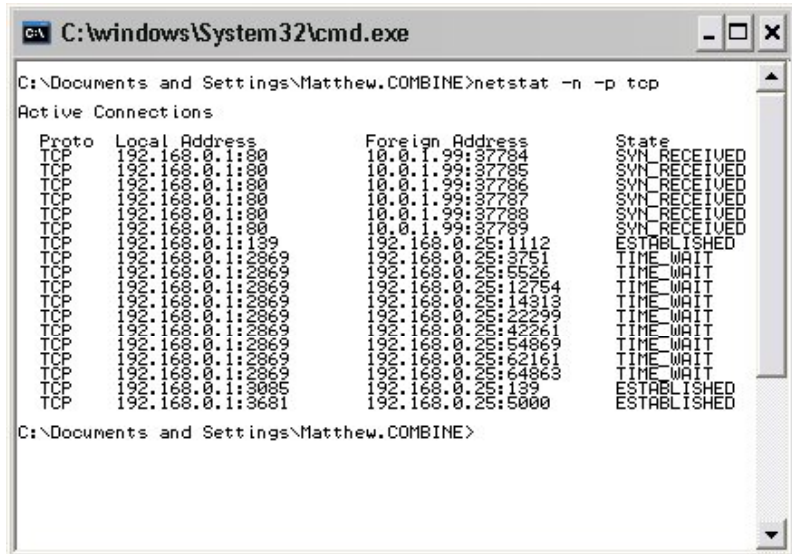
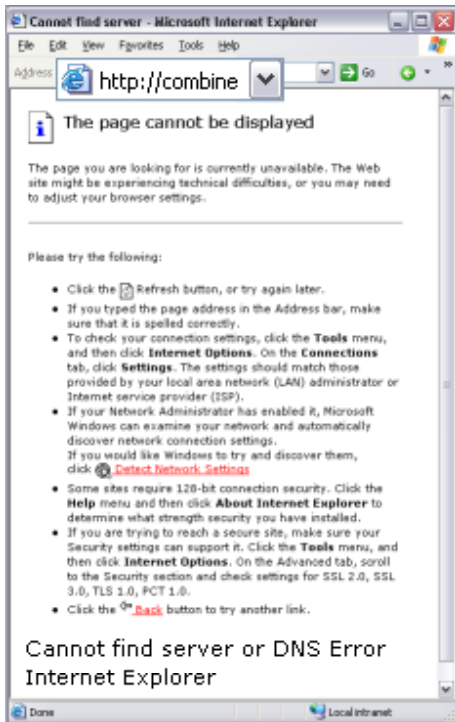


In this example the unreachable source IP address is 10.0.0.1 and the victims IP address is 192.168.0.25, (port 80 is attacked)

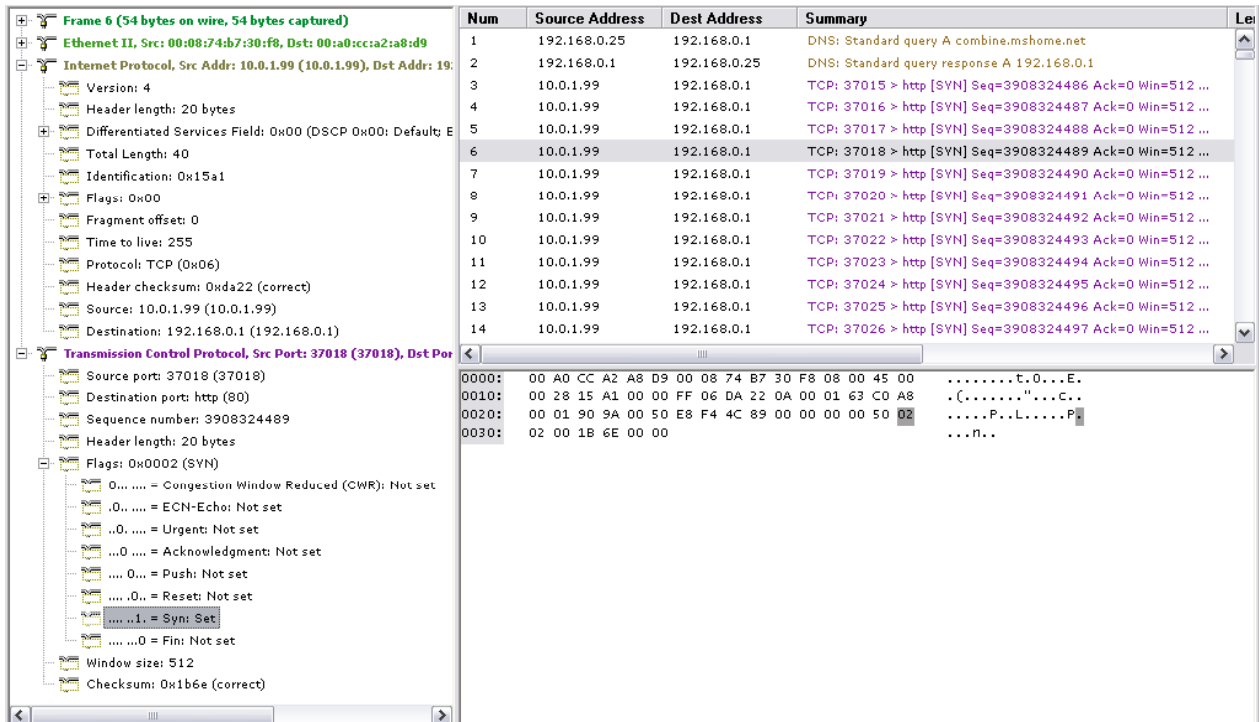
5.1.3 Simulation

A simulation was performed to demonstrate the effects of a SYN flood attack. The attack is launched from the HAL computer on a small LAN network (discussed in chapter 4.2 ^{page 20}) The victim machine (combine) is running the Apache http web server ^[11] on port 80. A sample of my own portfolio website is hosted on this server. (<http://hutchinson.ijug.org>) Under normal operation and using a web browser on HAL, a request to <http://combine> or <http://192.168.0.1> returns this site to the browser. A FLOODSLEEP value of 100 ms was used to represent Internet packet transfer delay on the 10 MB/s LAN.

The following screenshots are taken from both combine (victim) and HAL (attacker) before, during and after the attack. Packetyzer was setup with a user defined filter to capture all TCP traffic to and from port 80 on both computers.



During the attack: Trying to access the URL http://combine from HAL (or combine) results in a 'Cannot find server' or DNS error. SYN_RECEIVED connection queue shown in NETSTAT tool on combine - Service to the website is denied



During the attack: Packetizer on HAL monitors each forged SYN packet sent to the victim. Packets for the DNS lookup of combine (before the attack starts) are also shown in the capture

- **Increase size of connection queue** – Although each vendors IP stack differs slightly, it is possible to adjust the size of the connection queue to help reduce the effects of a SYN flood attack. This is helpful, but not an optimal solution since it uses additional system resources.
- **SYN cookies** - This method attempts to eliminate the need to store incomplete connection information by including a package of information, or a ‘cookie’ in the SYN/ACK packet sent by the receiver to the originator. When the originator responds with the ACK packet, the cookie is returned and the receiver is able to extract the information needed to rebuild the connection. This allows legitimate users to connect, even under heavy attacks with the same unreachable source IP.

Most modern Linux, UNIX and BSD operating systems use encrypted SYN cookies. Windows NT 4.0 and later versions employ a dynamic backlog mechanism. When the connection queue drops below a preconfigured threshold, the system will automatically allocate additional resources. Thus the connection queue is never exhausted. See Microsoft knowledge base article Q142641^[12] for more information.

5.1.5 Random SYN flooder

There are many variations on the standard SYN flooding program. Each tries to counteract efforts made by security experts in preventing SYN floods. This is usually the case with any well known exploit; an ongoing ‘battle’ ensues, between the attackers and the network security experts. To overcome SYN cookies, I designed a program that assigned random source IP addresses in the SYN packets. Each packet arriving at the victim is then considered to originate from a different unreachable host. Appendix B lists the source code for this program.

The code is similar to the standard SYN flood attack with a few exceptions. The main method no longer requires an unreachable source IP from the command line. This address will be randomly generated before sending each SYN packet, with a call to generateRandomIP(). Again the program is executed from the command line, the argument usage is;

```
RandSynflooder <target-host> {port-num} {num-times}    e.g. RandSynflooder yahoo.com 80 2000
```

```
string generateRandomIP(void)
```

This method returns a random source IP string of the form 129.62.x.y where $0 < x < 255$ and $0 < y < 255$. The first two quads in the IP address are 129.62; this is part of the unreachable address. X and Y make up the full address. The generated IP is not resolved (since it is unreachable) and is assigned to srcIP before constructing and sending each SYN packet.

The source port number is also randomised for every packet with a call to generatePort(). This method returns a random unsigned integer in the range 1 – 65535. To further randomise packet information, the

5.1.6 Attack analysis

Even with a very small amount of network traffic SYN flooding is very effective. Many other DoS attacks such as ICMP ECHO floods and bandwidth consumption attacks require the attacker to transmit huge amounts of network traffic. Not only does this make the attack more noticeable by decreasing the amount of available bandwidth, but it also adds to the general traffic problems of the Internet. SYN flooding can be extremely effective while sending as little as 360 packets / hour. This means an attacker with a connection rate as low as 28.8 KB/s can disrupt traffic to a high speed network service provider.

SYN flooding is also a stealth attack, since the source SYN packet contains a spoofed IP address. Tracing the origin of an attack is almost impossible (without contacting ISPs directly and scanning hundreds of log files.)

Today's DoS scenario barely resembles the earlier approaches taken against Panix.com in 1996. By 1999, attacks had become more sophisticated. Examples include distributed DoS attacks, attacks on network infrastructure and the use of high speed networks. As a result a large number of countermeasures have been put in place to detect, trace and stop SYN flood attacks. Although no longer devastating, the attack can still deny service for a period of time.

5.2 SynDetector

SYN flood countermeasures are readily available, and in some cases built into new operating systems e.g. the dynamic backlog algorithm (XP) and SYN cookies (Linux). Using these mechanisms it is not always obvious (to the user) that a SYN flood attack is in progress. SynDetector was written to detect and warn against a possible SYN flood attack. The application is designed to run in the background while the user hosts a service across a network (LAN, or Internet.) The program is not a SYN flood countermeasure since it does not prevent the attack from causing disruption. However it does identify and track possible SYN floods. This advanced warning allows the user to take the appropriate action on systems with no countermeasures (i.e. disabling a port or blocking an IP address).

The application has been written for Windows using Visual C++. Again the program makes use of the Komodia library ^[7], particularly the CSniffSocket class. A description of the application design and source code now follows.

5.2.1 Application Design

The SynDetector program will run in a Windows dialog box, and make use of the Microsoft Foundation Classes (MFC) to provide a standard graphical user interface (GUI). The basic requirements of the application are to;

- Allow the user to choose a network interface to monitor
- Monitor all incoming TCP SYN packets on this interface
- Display information on each SYN packet received
- Identify and count any TCP SYN packets that use an unreachable source address
- Issue a warning when a maximum flood count value is reached
- Identify which port has been targeted by the attack

By sub-classing the Komodia CSniffSocket class a new custom 'ClientSocket' class can be used to monitor or 'sniff' all incoming and outgoing packets on any network interface. Using the OnSocketRecieve() method a built-in program packet filter, can look for any incoming TCP packets that have the SYN flag set. The source IP address from such a packet can then be extracted from the IP header. An ICMP Ping request on this IP can test if the address is Alive (reachable), or unreachable (or invalid). The CPingSocket class in the KomodiaIPUtils library provides this functionality.

If the source IP address is reachable, ICMP Ping reply information will be displayed and the program will continue 'sniffing' for incoming SYN packets. If the source IP is found to be unreachable, a 'flood'

counter will be incremented. Once this counter reaches a maximum value, a warning will be issued to the user stating the SYN packet's unreachable source IP and targeted destination port.

SynDetector assumes that a maximum number of SYN packets arriving with an unreachable source address qualify as a possible SYN flood attack, even if different port numbers are targeted. With the maximum flood value set to 25-50, this is a fair assumption. Most SYN flood attacks typically send over 50 SYN packets before any real disruption is caused.

The application should have a user friendly GUI to allow the user to select an interface, and show information on SYN packets as they arrive. A 'Clear Data' button should also be available, to reset the flood count and clear the data window. The following code walkthrough explains how this application was implemented.

5.2.2 Code Walkthrough

The complete source code and header files for this application are included in Appendix C (excluding Visual C++ generated files). The code is fully commented and each method is listed with a short description (name, arguments and purpose). The main class files, ClientSocket.cpp and SynDetectorDlg.cpp will now be discussed. Other supporting files are used as GUI resources and message references in MFC functions. They are created automatically during GUI building using the MFC template wizard in Visual C++.

SynDetectorDlg.cpp

This file contains all the class descriptions relating to GUI construction and event handling. It also creates a new ClientSocket class, and binds it to a default interface before starting to monitor packets. The file contains some methods and class definitions inserted by the MFC template wizard during GUI design. These methods will not be discussed since they are automatically generated and have not been altered. Other smaller methods that are self-explanatory and related to GUI hooks will also be omitted. The header file SynDetectorDlg.h contains the class definition and function prototypes.

```
BOOL CSynDetectorDlg::OnInitDialog()
```

This method constructs the main dialog window and initialises the ClientSocket object. The method first assigns an 'About' menu item and icon to the dialog window. It then tries to initialise the TCPIPLibv4 for socket communication by calling CSpoofBase::InitializeSockets(). If no errors occur during initialisation a new ClientSocket object is created (m_socket). The ClientSocket object is passed a reference to the dialog window m_dataList, and m_floodCount components during construction. This allows the ClientSocket to update the m_dataList window (showing SYN packet information) and the m_floodCount text box (showing current flood count) during runtime.

The `BuildInterfaceList()` method is then called, populating the interface list box with any network interfaces found on the host machine. Calling `OnSniff()` starts the `SynDetector`, and packet monitoring begins.

```
CSynDetectorDlg::OnDestroy()
```

`OnDestroy()` is called when the application exits, or the dialog window is closed. It frees all resources used by the GUI, and closes any open socket connections with a call to `CSpoofBase::ShutdownSockets()`. The `ClientSocket` object (`m_socket`) is also deleted.

```
BOOL CSynDetectorDlg::BuildInterfaceList()
```

This method uses the `CInterfaces` class (provided by the Komodia library ^[7]) to examine all network interfaces available on the computer. A `CInterfaces` object is created (`pInter`) and a recursive call to `GetInterfaces()` populates the `m_InterfaceList` list box with the corresponding IP addresses. A default interface is then selected, so when `OnSniff()` is called by `OnInitDialog()` the program starts monitoring traffic on this interface. The `pInter` object is deleted after use and the method returns `FALSE` if any error occurs.

```
CSynDetectorDlg::OnSniff()
```

`OnSniff()` is first called during program initialisation from the `OnInitDialog()` method. The 'Select' button also calls this method when pressed. Meaning the user can change which interface to detect SYN packets on during runtime. `OnSniff()` begins by examining the `m_InterfaceList` list box and getting the selected IP address. It then binds the `ClientSocket` object (`m_Socket`) to this address. A call to `m_Socket->Sniff(TRUE)` starts packet monitoring. Incoming or outgoing packets will generate a call to the `CClientSocket::OnSocketReceive()` method (in `ClientSocket.cpp`).

```
CSynDetectorDlg::OnClear()
```

This method is called when the user clicks the 'Clear Data' button. A call to `m_Socket->ResetFloodCount()` resets the static flood count value to zero in the `ClientSocket` class, and updates the dialog window to show this. `OnClear()` then deletes all text in the `m_dataList` window (if there is any).

ClientSocket.cpp

This file contains the `ClientSocket` class implementation. The header file `ClientSocket.h` contains the class definition and function prototypes. `ClientSocket` extends directly from the `CSniffSocket` class provided by the Komodia ^[7] library. `CSniffSocket` allows packet monitoring (incoming and outgoing) on any network interface. This `ClientSocket` class is essentially a simple 'incoming' packet filter. The class contains one static unsigned int variable `synFloodCount`. This is used to track the number of SYN packets with unreachable source IP's. A `MAXNUMSYNS` definition is used as a limit on the `synFloodCount`. When `synFloodCount` reaches `MAXNUMSYNS`, a warning is displayed to the user.

This class also makes use of the CPingSocket class. Again provided by Komodia, CPingSocket allows the sending and receiving of standard ICMP ECHO requests and replies. This is used to determine if a source address is reachable. The CPingSocket must be used in the context of a 'message loop'. A message loop is initiated when a Ping request is sent. The loop continues executing a method repeatedly until it returns TRUE. This allows the program to wait for a Ping reply. Each method in the ClientSocket class will now be discussed in more detail.

```
CClientSocket::CClientSocket(CListBox*, CEdit*)
```

The constructor for the ClientSocket class takes two arguments. They provide a reference to GUI components on the main SynDetector dialog, allowing ClientSocket methods to reference the data list window (m_dataList) and the flood counter edit box (m_floodCount).

```
BOOL ShouldStop()
```

This is the method that is executed in the Ping message loop. The message loop continues forever until this method returns TRUE. It is used after a Ping request has been sent. ShouldStop() checks the global Ping socket object (gpingSocket) using IsPingDone(), to determine if the Ping has timed out or replied successfully.

```
BOOL CClientSocket::Bind(const std::string&, unsigned short)
```

Bind() is used to bind the ClientSocket to a specific interface. It is called by the SynDetectorDlg::OnSniff() method. A CSniffSocket can be bound to an IP address and port number. If no port number is specified, the Socket will 'sniff' all ports on the bound interface. The bound interface IP address is saved and referred to as the source address (m_sAddress). IP packets with this source address are considered outgoing packets, and with this destination address; incoming packets.

```
BOOL CClientSocket::OnSocketReceive(int)
```

This method contains all packet filtering activity. OnSocketReceive() is called every time a packet (incoming or outgoing) is injected on the network interface bound to this socket. A character buffer is used to store all packet data. The method first determines if the packet is incoming by examining the IP header destination address. If the packet is using the TCP protocol a call to AnalyzeTCP() is made. This method checks for a SYN packet (TCP flags 0x000010 = 2). If found it returns the TCP destination Port (or -1 if not found).

With this information, the method displays SYN packet details on the dialog window and tries to send a Ping request to the source IP. If the Ping request completes successfully, ICMP Ping reply information is displayed.

If the Ping request cannot be sent, or the reply times out, synFloodCount will be incremented and the data list window will display the Ping error.

The new flood count value is updated on the main dialog window and a check is made to ensure synFloodCount has not reached MAXNUMSYNS. If it has, a warning message is displayed, explaining the number of SYN packets, the source IP address and targeted port number. The warning suggests disabling this port until the attack subsides. The method then returns and packet monitoring on this socket continues.

```
int CClientSocket::AnalyzeTCP(char *)
```

AnalyzeTCP() is used to check for a TCP SYN packet. The method is called with a reference pointer to TCP header data. This is used to construct a TCPheader object. If the SYN flag has been set, i.e. 0x000010 = 2, then the method returns with the TCP destination port. If this is not a TCP SYN packet the method returns -1 (an invalid TCP port number).

```
int CClientSocket::ResetFloodCount()
```

ResetFloodCount() is called from the SynDetectorDlg::OnClear() method, and is used to zero the synFloodCount variable. The main dialog window is updated to reflect this.

5.2.3 Simulation

Both the RandSynflooder and Synflooder attacks will be launched on the same test network used in previous simulations. This time the SynDetector application will be running on combine (the victim). The program should detect and warn against both attacks, since MAXNUMSYNS has been set to 25. This simulation illustrates the effectiveness of the SynDetector application. The figure below shows SynDetector reacting to a few *valid* SYN packets (i.e. a valid source IP address, 192.168.0.196 from HAL). ICMP Ping information is displayed in each case.

Figure 5.6 – SynDetector and valid SYN packets

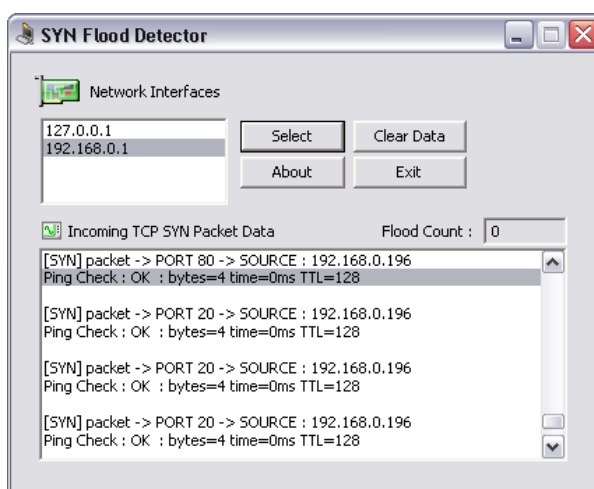
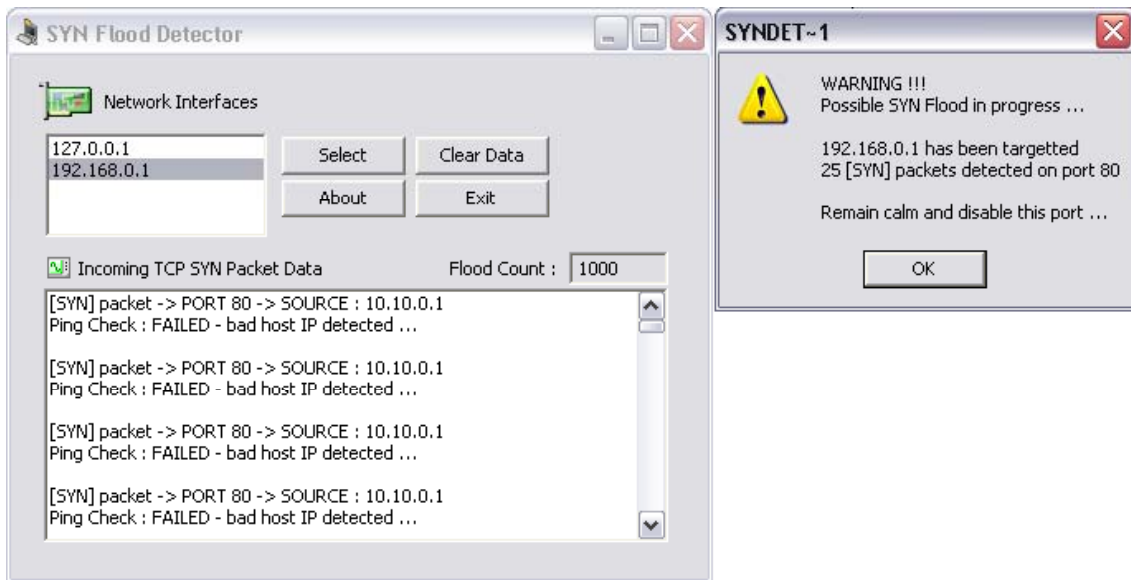


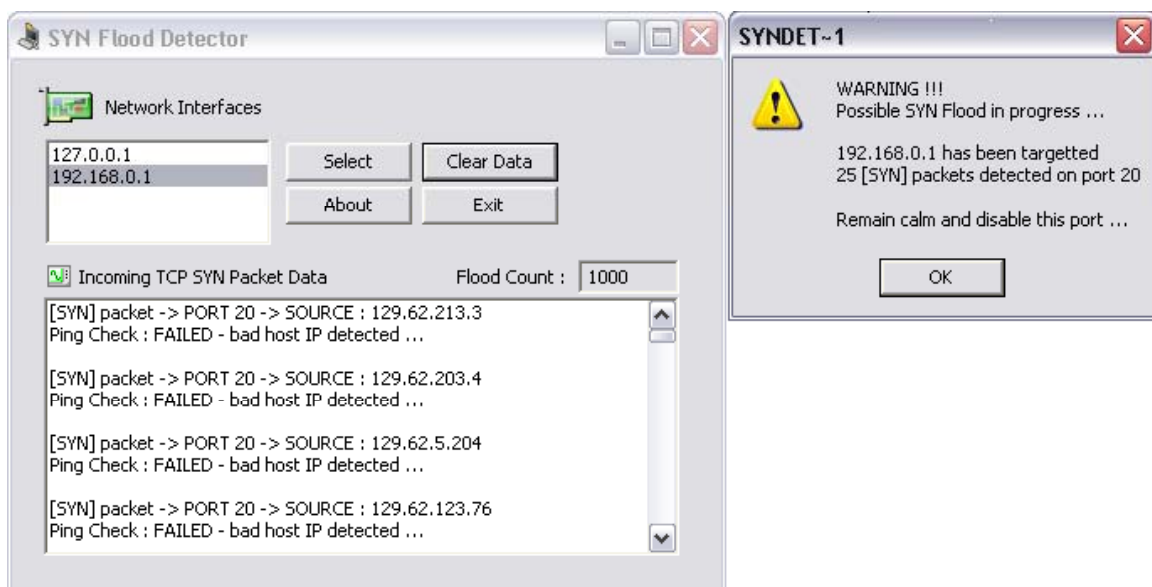
Figure 5.7 shows the application under attack from a standard SYN flood (Synflooder.exe). The source IP in each packet is identical and unreachable. 1000 spoofed SYN packets are sent to port 80 on the victim machine. Trying to Ping the address fails and the flood count soon reaches its maximum value, (25) displaying the warning message.

Figure 5.7 – SynDetector under a SYN flood attack



The figure below shows SynDetector running on combine (the victim) during a random SYN flood attack (RandSynflooder.exe). This time 1000 spoofed SYN packets are directed at port 20 on the victim. The TCP data window shows the spoofed source address for each SYN packet received. Again a warning is raised after MAXNUMSYNS is reached.

Figure 5.8 – SynDetector under a Random SYN flood attack



The SynDetector provides an early warning mechanism against SYN flood attacks. On receiving a warning message the user is immediately aware of which port the attack is targeting. If a standard SYN flood is in progress, the user can block the unreachable IP address and still provide a service across the network to legitimate clients. If a random SYN flood is in progress the user can disable the targeted port number and wait until the attack subsides. The data window provides useful information on TCP SYN packets allowing the user to track the flood. Future modifications to SynDetector are discussed in the Conclusions - Future Work chapter.

The application does not prevent a SYN flood attack from denying service. The real problem can only be solved by securing vulnerabilities in the IP protocol. Under existing Internet architecture, preventing IP spoofing is impossible. SYN cookies and the dynamic backlog mechanism are work-around methods, only a new IP protocol (e.g. version 6) will offer a perfect solution.

5.3 SMURF

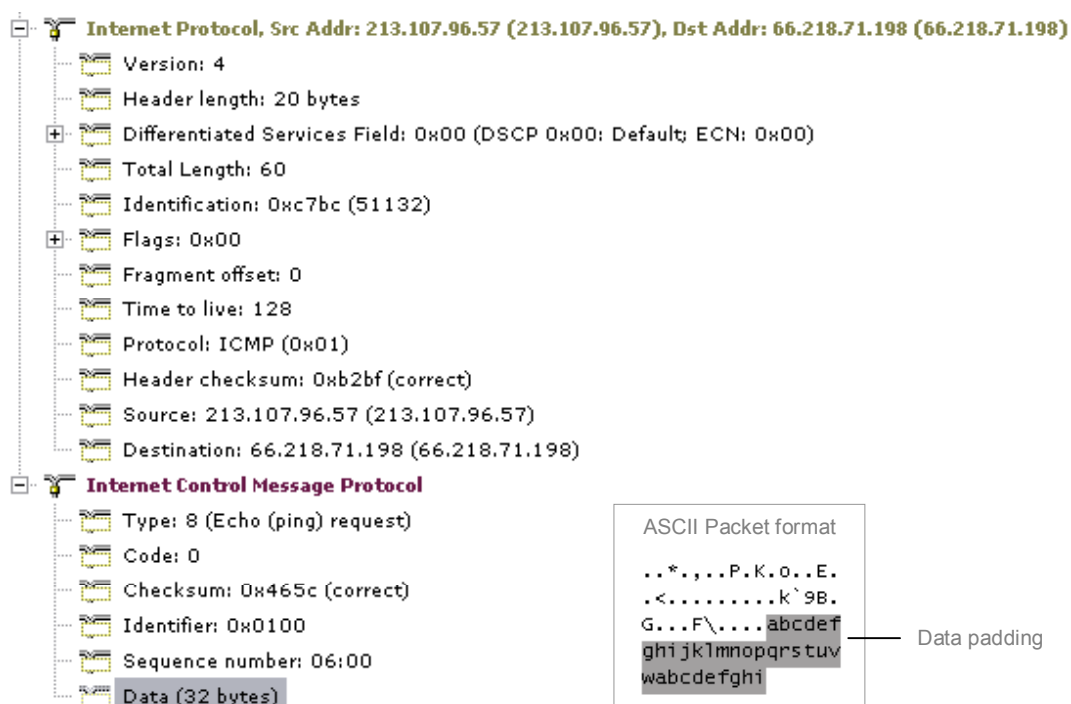
SMURF is one of the most devastating types of DoS attack. It makes use of a technique known as ‘traffic amplification’ to target network-level hosts. Attackers can *amplify* their DoS attack by engaging multiple sites to flood the victim’s network. Using this process attackers with a slow 22 KB/s connection can completely saturate a T3 (45 MB/s) connection. The attacker must ‘convince’ the amplifying systems to send traffic to the victim’s network, and in the case of SMURF, this is done by taking advantage of poor security in ICMP (Internet Control Message Protocol).

SMURF attacks began to surface around January 2000, first targeting small IRC (Internet Relay Chat) servers, and eventually hitting the network infrastructure of some large commercial sites including CNET.com. However, this attack is not limited to public web servers or network services; it is possible to target routers, switches or any other network equipment with an IP address. The attack is not to be confused with the ICMP Ping of Death, which involves manipulating the maximum size of an ECHO packet.

5.3.1 Attack concept

The SMURF attack takes advantage of the ICMP protocol and directed-broadcasts. It requires a minimum of three actors, namely the attacker, the amplifying network and the victim. A typical ICMP echo packet is shown below. It consists of an IP header and ICMP header. This packet was generated by sending a valid ping request to yahoo.com and capturing it in Packetyzer.

Figure 5.9 – ICMP echo request packet



The echo reply packet has exactly the same structure, values that change are the checksums and *Type* field in the ICMP header (0 indicates an echo reply). The source and destination address in the IP header are simply reversed. The data in both packets serves as padding and varies so the packet meets the minimum required size before transmission. In this case it is 32 bytes of filler data (i.e. *abcdef....* etc.)

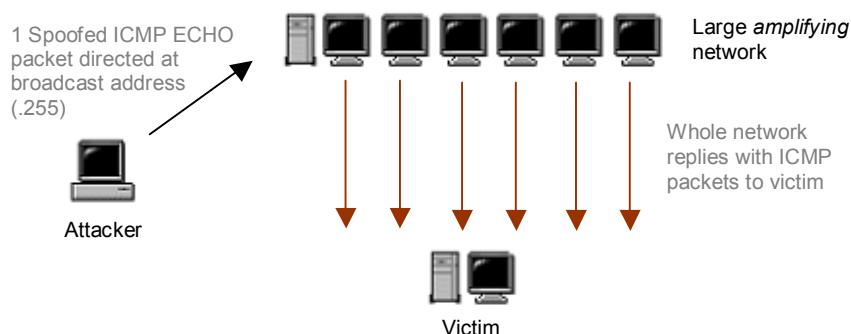
An attack begins by sending a few spoofed ICMP echo packets to the broadcast address (.255) of the amplifying network. Direct broadcast requests are typically used for diagnostic purposes, to see what is alive in the network without using ‘Ping’ on each address in the range. The source address field in these packets is forged to make it appear that the victim has initiated the request.

Because the ICMP ECHO was sent to the broadcast address, *all* machines on the amplifying network will respond to the victim. For example a single ICMP ECHO packet sent to an amplifying network of 100 machines, effectively allows the attacker to amplify the DoS attack by 100. The attacker must therefore find a large network which, when replying to a spoofed ICMP ECHO packet, will completely saturate the victim. Two parties are disrupted during this attack, the intermediary broadcast devices (or amplifiers), and the spoofed address target (victim). Thus the victim is the target of a large amount of ICMP traffic that the amplifiers generate.

A variant of this attack known as *fraggle* operates under the same principle, using spoofed UDP traffic directed at port 7 (ECHO). Each system on the amplifying network that has ECHO enabled will respond back to the victim’s host, creating large amounts of traffic. Even if ECHO is not enabled, an ICMP unreachable message will be generated and sent, still consuming bandwidth. Fraggle is a simple re-write of SMURF.

Attackers can go one step further and send spoofed ICMP packets to more than one large amplifying network. With only a few packets the attacker can levy a massive amount of traffic on the victim. Since network resources are consumed, SMURF and fraggle are considered a form of bandwidth consumption attack. Figure 5.10 shows a typical SMURF attack.

Figure 5.10 – SMURF attack with amplification



5.3.2 Example Scenario

Due to the fact that SMURF and fraggle attacks involve compromising a large number of machines to act as an amplifying network, it would be impractical to simulate in a small lab, and illegal to test over a public network. To illustrate the attack consider this example.

Assume a co-location switched network with 100 hosts, and an attacker with access to a T1 line. The attacker sends, a 768 KB/s stream of ICMP ECHO (Ping) packets, with a spoofed source address of the victim, to the broadcast address of the 'bounce site'. These Ping packets hit the bounce site's broadcast network of 100 hosts; each of them takes the packet and responds to it, creating 100 Ping replies outbound. Thus 76.8 MB/s is used outbound from the "bounce site" after the traffic is multiplied. This is then sent to a T3 connected victim, (the spoofed source of the originating packets). Even with this T3 connection (44.736 MB/s) the malicious traffic is enough to starve the victim of its network resources.

5.3.3 Countermeasures

This attack relies on the router serving a large multi-access broadcast network to frame an IP broadcast address (such as 10.255.255.255) into a layer 2 broadcast frame (for Ethernet, FF:FF:FF:FF:FF:FF). RFC 1812, ^[6] "Requirements for IP Version 4 Routers", specifies:

“A router MAY have an option to disable receiving network-prefix-directed broadcasts on an interface and MUST have an option to disable forwarding network-prefix-directed broadcasts. These options MUST default to permit receiving and forwarding network-prefix-directed broadcasts.”

With IP providers and IP applications as we know them today, this behaviour should be not needed, and it is recommended that directed-broadcasts be turned off, to suppress the effects of this attack. In most modern routers (Cisco etc.) this function is disabled by default. UDP traffic at border routers should also be limited to only necessary systems, in the event of a fraggle attack.

To protect a victim from SMURF traffic, it is possible to configure some operating systems to silently ignore all ICMP ECHO reply packets (hence disabling diagnostic applications such as Ping). In order to trace an attack, the victim must work closely with the amplifying site. By systematically reviewing each router, starting with the amplifying site and working upstream it may be possible to trace the origin. This is accomplished by determining the interface at which the spoofed packet was received, and then tracing backwards.

PowerTech Information Systems in Norway, host a SMURF amplifier registry (SAR) website ^[13]. The site lists all known SMURF amplifiers and is updated every five minutes. The SAR lets you probe

Internet connected IP networks to see whether or not they are configured in a way that will allow perpetrators to use them for SMURF amplification. A full dump of the SAR is available at any time. The information can be used to block traffic to or from the listed networks, or to help in contacting the networks, to make them understand how important it is to prevent SMURF amplification.

5.3.4 Attack analysis

SMURF attacks are clearly an effective DoS tool. Again they allow a form of stealth attack, since the victim is not directly aware of the origin of ICMP traffic. The amplification mechanism allows a low speed connected attacker, to generate huge amounts of traffic. And intermediary amplifiers have no knowledge their network is being used to launch an attack until it is too late.

Sites like PowerTech's SAR registry ^[13] are helping to solve the problem. Unfortunately new 'user-friendly' attacking tools are emerging with SMURF and fraggle built in with a whole suite of common DoS attacks.

6 DNS Attacks

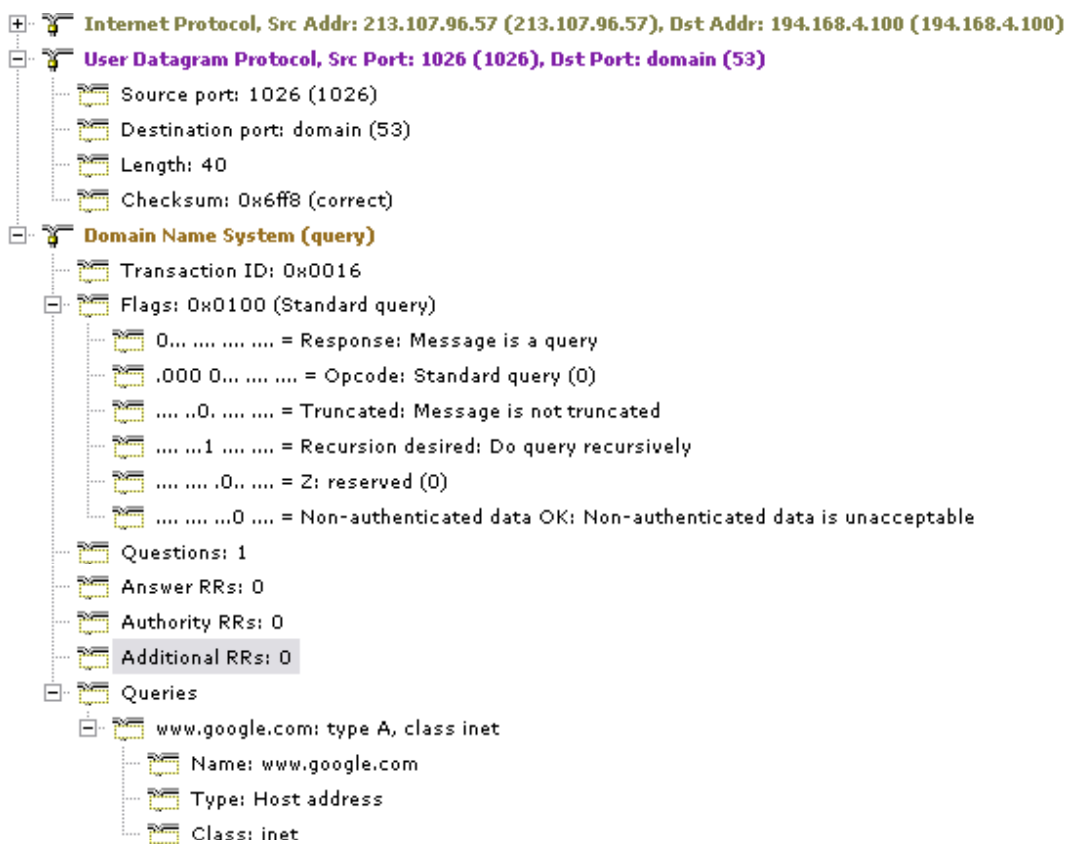
6.1 Background Information

The basic function of the Domain Name System (DNS) is to translate a domain name, which is easy for humans to remember, into a numerical IP address which is simpler for computers. It is a large distributed database in which local administrators have control over segments that contain the information about their domain.

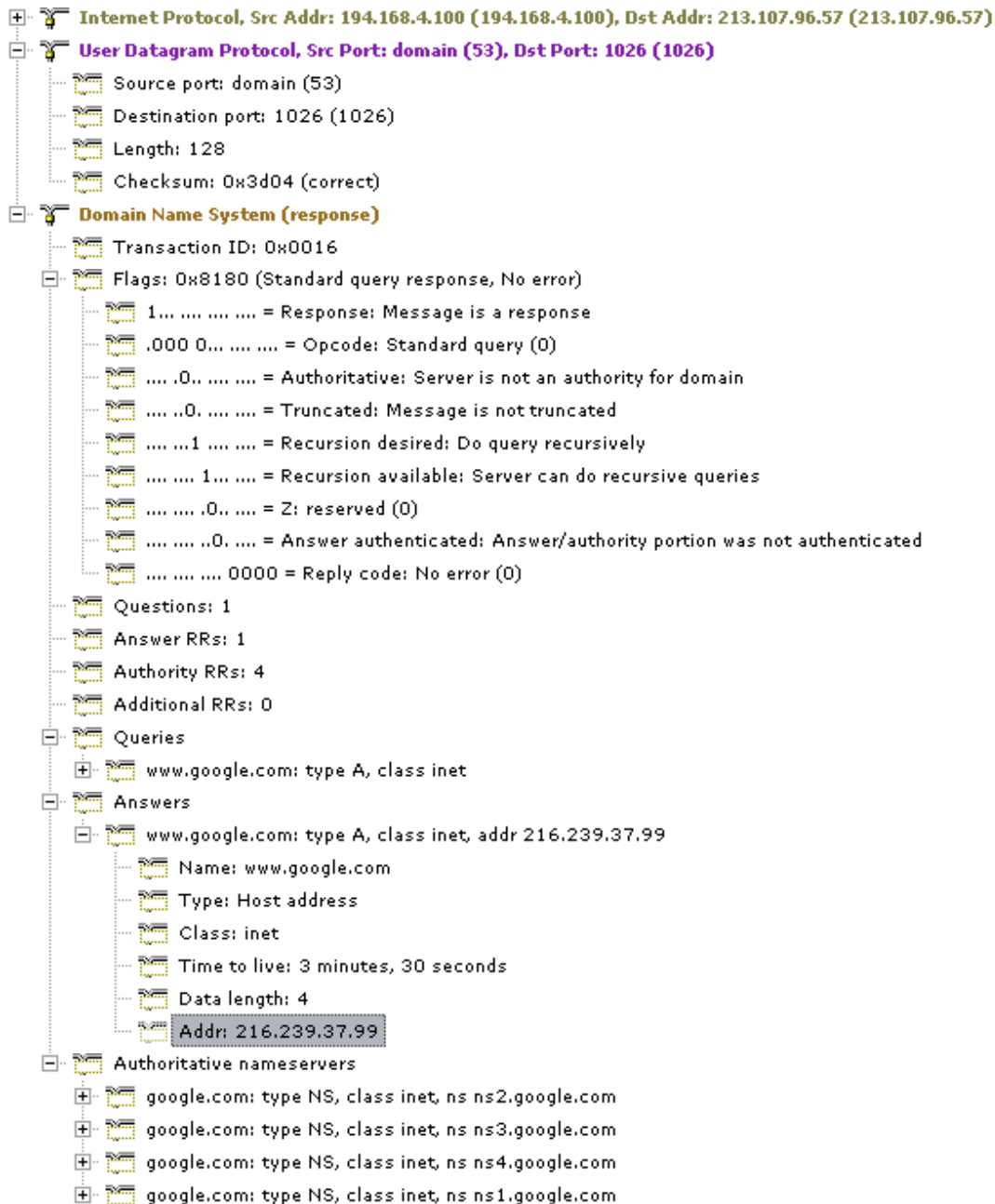
Name servers are the software programs that implement the database and respond to queries from clients. Clients can be either a host's resolver or another name server. The DNS protocol (RFC 1034^[6]) defines the message format a client uses to query the name server database and the format of the name server's response.

DNS uses the UDP protocol for resource records (faster packet transfer between hosts). The figure below shows a typical DNS packet resolving the IP address of google.com from a name server (194.168.4.100 – port 53). The standard response to this request is also shown (captured with Packetyzer).

Figure 6.1 – DNS packet analysis



Standard DNS query to resolve IP for **www.google.com**



- Standard DNS response in resolving IP for www.google.com - address is **216.239.37.99**

When a client wishes to know the IP address of a particular domain it sends a query to the name server. If the name server is authoritative for that domain name, (in which case it will always know answer) or if it contains the information in its cache, it will answer the query immediately.

If the name server is not authoritative for the domain it may still be resolved, since most resolvers always send recursive queries. As long as the name server is configured to accept them, it can recursively resolve the domain name. The originally queried server now becomes the client of the authoritative name server. When it receives the answer from the authoritative name server, the original name server passes it on to the resolver. A cached copy of the response is also stored for a limited period of time. When subsequent

queries are made for the same domain name it will use the information in its cache to generate a response, instead of doing more recursion. In the above example query, there were 4 authoritative servers for google.com (ns2.google.com, ns3.google.com etc.) Only one server responded with an answer containing a valid IP address. If one 'google' name server is down, the IP address can still be obtained from another. If the domain name doesn't exist or the name server is configured to refuse recursive queries, an error will be returned to the resolver.

DNS is critical infrastructure in the Internet's framework. Without it, everyday Internet use would be difficult, if not impossible. The US Government has acknowledged this and has urged federal dollars to be spent on DNS related security research ^[14]. A DNS attack could invoke two possible scenarios, depending on where the attack was targeted.

Local attack

A local DNS attack could occur if a company's DNS servers were taken out of commission. Internet users would be unable to resolve any of the website domain names into IP addresses and therefore, the company's online presence would cease to exist. No one would be able to browse the site or email technical support for help. This is exactly what happened to Microsoft in January 2001 when its DNS servers were taken offline for more than 24 hours ^[15]. All of microsoft.com was unavailable to the rest of the Internet for this duration.

Global attack

The DNS database can be thought of as a huge, inverted, hierarchical tree. At the top of this tree, are 13 roots (the so-called root name servers). These 13 servers delegate information for all of the top-level domains (com, net, org etc.) This means that they contain information on how to find out which DNS server is authoritative for nearly every domain on the Internet. Anytime a DNS server needs to look up an address it doesn't have cached, it will eventually consult one of the 13 root name servers.

If all of these 13 root name servers were attacked and disabled, DNS would not function on a global scale, thus disabling the Internet entirely. In October 2002, a distributed denial of service attack (DDoS) managed to disable 9 of the 13 name servers for a period of time. The attack was identified and stopped before any major damage was caused. The Washington Post called it the 'largest and most sophisticated attack ever attempted on Internet infrastructure' ^[16].

Since DNS is critical in every aspect of Internet communications, (email, www, ftp, IRC etc.) It is surprising to find that the protocol has very poor methods of authentication. BIND (Berkley Internet Name Domain) is the largest and most widely implemented DNS package available. It is used almost universally by anyone installing a DNS server. BIND was developed in 1988 and has undergone many revisions. With each increase in code complexity BIND presented the opportunity for more vulnerabilities

and errors to appear. Attackers have exploited this fact and have managed to create some complex DoS attacks. These attack mechanisms will now be discussed.

6.2 PTR Record Spoofing

A PTR ('pointer') record is used with reverse maps, mapping IP addresses to host names. It is used by name servers to verify that an IP address does actually correspond to the host name stored in an authoritative name server database. When recursion is enabled on vulnerable versions of BIND (earlier than v4.95) an attacker can poison the cache of the name server. Thus the IP address for a hostname can be altered in the cache by the attacker. Although it is possible for the attacker to direct traffic to a different (but valid) IP, it is also possible to perform a DoS attack by directing traffic to a non-existent (unreachable) IP. When users of the vulnerable name server wish to go to www.victim.com they will never receive an answer from the unreachable address, effectively denying service to www.victim.com. PTR record spoofing is also known as DNS cache (or information) poisoning – discussed in chapter 2 (Types of attack ^{page 7})

This cache poisoning is a 'man-in-the-middle' attack, since the attacker is providing false information to the victim, who assumes they are communicating with a trusted domain name server. The poisoning is possible because the attacker can easily predict a BIND DNS transaction ID. Transaction ID's are used in BIND to identify a DNS request with a DNS reply. This ID is the *sole* form of authentication for DNS. The attacker predicts the transaction ID through a 'brute-force' method, sending thousands of spoofed queries followed by spoofed replies to the name server. Each query and reply tries a different transaction ID until a valid one is guessed. To solve this, all new versions of BIND were updated to use random transaction IDs.

6.3 The Birthday Attack

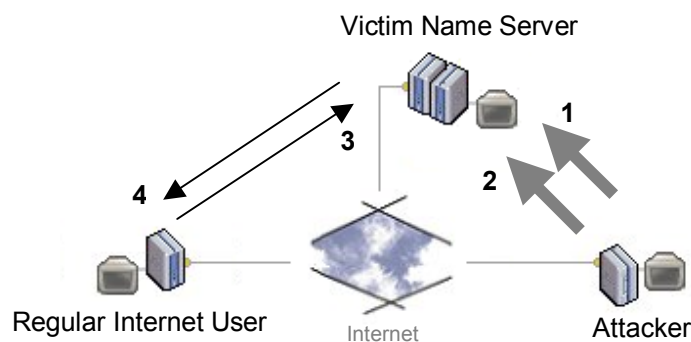
Attackers faced a new challenge and began to develop an alternative mechanism for DNS cache poisoning. In late 2002 a security advisory ^[17] was released documenting a new problem with BIND. A flaw exists in the BIND software that generates multiple queries for the same domain name at the same time. Because of this attackers encounter improved odds of guessing the correct transaction ID. This is the classic 'Birthday Attack', which is derived from the 'Birthday Paradox', described below:

"A birthday attack is a name used to refer to a class of brute-force attacks. It gets its name from the surprising result that the probability that two or more people in a group of 23 share the same birthday is greater than ½. Such a result is called a birthday paradox. If some function, when supplied with a random input, returns one of k equally-likely values, then by repeatedly evaluating the function for different inputs, we expect to obtain the same output after about $1.2k^{1/2}$. For the above birthday paradox (days in a year), replace k with 365. " ^[18]

This same theory can be applied to pseudo-random numbers, such as those generated by BIND for DNS transaction IDs. IDs are single 16bit numbers and the chance of guessing an ID (without using the Birthday Paradox) is 1 in 65535. A birthday attack nears 100% success after 700 queries have been sent. Using conventional methods, 700 queries would only achieve a $700/65535$ success rate (1.07%) Hence the attacker can now send a few hundred spoofed queries, rather than tens of thousands.

The Birthday attack shows that even a random number generator is vulnerable, if it generates multiple numbers for the same transaction. A typical birthday attack scenario is shown below. The attacker must first work out the source port for a recursive query. This can be found out by issuing a request for a DNS lookup of a hostname on the attacker's server. When a reply is received the packet can be examined and the source port noted. This source port is then used when sending the spoofed DNS replies (step 2). Since BIND does not randomise its source ports, the attacker can take this as valid and use it in subsequent queries.

Figure 6.2 – Birthday Attack Scenario



1. Attacker sends a large number of DNS lookup queries to the victim name server, all for the same domain name (e.g. www.example.com) This name server is not authoritative for this domain and must perform recursion - Therefore the victim name server requests a response from an authoritative name server for this domain (the race is on ...)
2. Attacker sends spoofed DNS replies giving fake answers for the requests just made. With a valid transaction ID reaching the name server first, the answer forces it to cache the fake IP address for www.example.com
3. At a later time another regular Internet user, browses to www.example.com, a DNS lookup is performed on the Victim Name Server.
4. The Victim name server finds a resource record in its cache for www.example.com and returns the spoofed address information. The regular internet user is directed to a non-existent IP (hence DoS), or a different IP (attacker's own website posing as www.example.com).

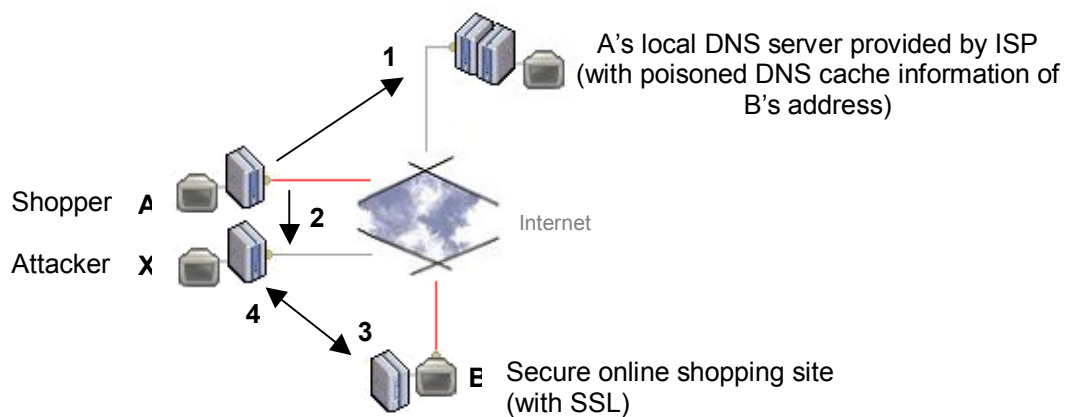
After step 1 the attacker must win the race between the first successful collision of the spoofed transactions, and the legitimate answer from the true authoritative name server. The attacker can gain an advantage in this race, by flooding the authoritative name server with ‘dummy’ queries (slowing down its response time.) The victim name server will cache the spoofed address for a period of time specified in the TTL field of the attacker’s response. This value will be set to maximum (by the attacker) to cause disruption for as long as possible.

The attacker may choose to redirect traffic for `www.example.com` to an IP address of their own. Some reasons for doing this include,

- Redirect a popular search engine to a pop-up ad site.
- Redirect a bank website to gain access to account passwords.
- Redirect an online shopping site and ask users to re-enter credit card details or other sensitive information. Then immediately forward these details to a generic email account.
- Redirect a news site to inject false stories and manipulate stocks.

Using the ‘man-in-the-middle’ property, cache poisoning can also be used to ‘sniff’ secure traffic between two hosts. The diagram below illustrates this scenario.

Figure 6.3 – Remote Traffic Interception



1. A wants to make an online purchase at B’s website. A requests a DNS lookup of B’s IP address.
2. The attacker (X) poisons the cache at A’s ISP, pointing traffic for B’s site to X’s system.
3. X accepts the incoming SSL connection, decrypts it, reads all the traffic, and makes the same request via SSL to B’s site.
4. Replies from B are read by X then sent back to A over the same encrypted session.
5. X now has A’s credit card number and any other sensitive information.

This is a powerful and ‘stealthy’ form of remote traffic sniffing. Both A and B have no knowledge traffic sent through the SSL (secure socket layer) connection has been intercepted. B’s website appears to function correctly, with response times only slightly longer from routing through the attackers system.

Summary

With any spoofed DNS attack there are two victims, the domain owner, who is losing traffic to their site, and the end user who gets redirected to the wrong IP address. Newer versions of BIND do not generate multiple queries for the same domain name, and are therefore resilient to earlier forms of the Birthday attack. The latest version also randomises its source port, providing further security against spoofed DNS responses. To counteract further attacks, all administrators should upgrade to the latest version of BIND. It is evident that hundreds of name servers across the Internet are still using old versions and remain vulnerable.

End users can insure they are using the correct IP address for a domain name by logging onto the ARIN WHOIS Database (www.arin.net). This is an online tool provided by the American Registry for Internet Numbers. It allows users to enter an IP address and the corresponding domain name is returned, verified and displayed with owner information.

6.4 DNS Bandwidth Consumption

An attacker can leverage other vulnerabilities in the DNS protocol to generate a large amount of traffic from DNS servers. This traffic is then used in a bandwidth consumption attack to starve network resources from a victim machine. Although it shares many of the same characteristics as Smurf, Fraggle and other attacks which use IP spoofing, it doesn’t utilize broadcast addresses to achieve amplification. Instead it uses the fact that DNS response messages may be substantially larger than DNS query messages.

Not including the IP or UDP headers, a query message is approximately 24 bytes (depending on the size of the query domain name). A response message could easily be double that size. This is illustrated in figure 6.1 ^{page 37}. The first packet capture shows a DNS query for www.google.com. The size of the entire packet is given in the ‘Packet size’ field of the Ethernet header, 74 bytes. The length of the UDP segment is 40 bytes which means the size of the DNS query message is 20 bytes (40 – 20). Compare this to the second figure showing a DNS response. It has a total packet size of 162 bytes of which the response message makes up 128 bytes – over three times the size of the original UDP query segment (40 bytes).

The attacker changes the source IP address to be that of a host on the victim network. Using the spoofed address, a DNS query for a valid resource record is crafted and sent to an intermediate name server. The

attacker will continually send the query to the intermediate name server with all the responses going to the victim network.

To increase the amplification effect the query is usually directed at a non-authoritative name server for the resource record. Thus the name server (as long as it is configured to) will do a recursive lookup to resolve the query. Recursive lookups add significantly to the amount of data returned in the DNS response message. Potentially, an attacker could consume the entire bandwidth of a T1 by generating just a few thousand responses.

DNSSEC

The only real prevention against any type of DNS attack is to modify the protocol to include a secure form of authentication. Internet security experts are currently developing a new protocol known as DNSSEC^[19]. It uses digital signing in all resource record queries. This authenticates the identity of the name server as well as validating responses. All name server responses include a previously generated digital signature of each resource record. However DNSSEC provides only one-way authentication. Only the responses from a name server are digitally signed, there is no authentication of the client. Thus DNSSEC does not prevent an attacker from using a name server for bandwidth consumption DoS attacks. The real vulnerability exploited in these attacks is IP spoofing.

The very latest version of BIND does provide support for DNSSEC. However large portions of existing name servers are still using an older version. It will take some time before DNSSEC can provide an effective solution, since every name server and client across the Internet would have to be updated.

7 DoS and Programming Flaws

Almost all vulnerabilities in computer software are caused by programming or design flaws. The searches for exploits in software have been the pursuit of hackers and crackers for years. Being able to identify and take full advantage of a programming flaw is considered highly skillful in the underground community. In some cases attackers have been able to gain root privileges, remote control or access sensitive information. Programming flaws make all these actions possible.

One common flaw that seems to reappear in any program is the failure to handle exceptional conditions properly. At the software design stage, programmers only consider a typical input data range. When unintended data is sent to a vulnerable program an exceptional condition may occur, sending the software into an unpredictable state. The consequences vary, from simply crashing the program to executing malicious code (after a buffer overrun).

For this reason many programs must be updated or ‘patched’ after their initial release. Attackers then look for new exploits and the cycle continues. Using programming flaws an attacker can remotely crash (or control) a network service on an un-patched or vulnerable system. To illustrate this, the IP fragmentation flaw (and its variants) will be discussed. Documenting all programming flaws in all network services is beyond the scope of this project.

7.1 Ping of Death

The Ping of Death is considered to be the original and oldest denial of service attack (next to SYN flooding). It makes use of a common and well known flaw in the Ping (ICMP ECHO request ^{page 40}) network diagnostic tool. It is possible to crash, reboot or otherwise kill a large number of systems by sending a Ping of a certain size from a remote machine.

IPv4 packets can be up to 65,535 ($2^{16}-1$) bytes long, including the header length (typically 20 bytes if no IP options are specified). Packets that are bigger than the maximum size the underlying layer can handle, are fragmented into smaller packets, which are then reassembled by the receiver. For Ethernet style devices, the MTU (maximum transmission unit) is typically 1500 bytes. An ICMP ECHO request contained in the IP packet, consists of eight bytes of ICMP header information, (RFC 792 ^[6]) followed by the number of data bytes in the ‘Ping’ request. Hence the maximum allowable size of the data area is $(65535 - 20 - 8) = 65507$ bytes.

Attackers realised that it was possible to send an illegal ECHO packet with more than 65507 bytes of data, due to the way fragmentation is performed. IP fragmentation relies on an offset value in each packet to determine where the individual fragment goes upon reassembly. Thus on the last fragment, it is possible

to combine a valid offset with a suitable fragment size such that $(\text{offset} + \text{size}) > 65535$. Typically most systems do not attempt to process a packet until all the fragments have been received. This opens the possibility for overflow of 16 bit internal variables, resulting in system crashes, protocol hangs, and other problems.

Using IP source address spoofing on ICMP packets, attackers can act anonymously. The attacker needs to know nothing about the victim machine other than its IP address. In 1996 at its most destructive stage, over eighteen major operating systems were found to be vulnerable including Windows 95 and Windows NT 3.51, 4.0. ^[20]

The Ping of Death was realised in many forms, also known as the long ICMP attack, it could be found under the program names, *jolt*, *sPing* and *IceNewk*. By the end of 1997, operating system vendors had released patches to combat the exploit. Still, many web sites continue to block incoming ICMP Ping messages to prevent any future variations of this attack.

7.2 IP Fragmentation

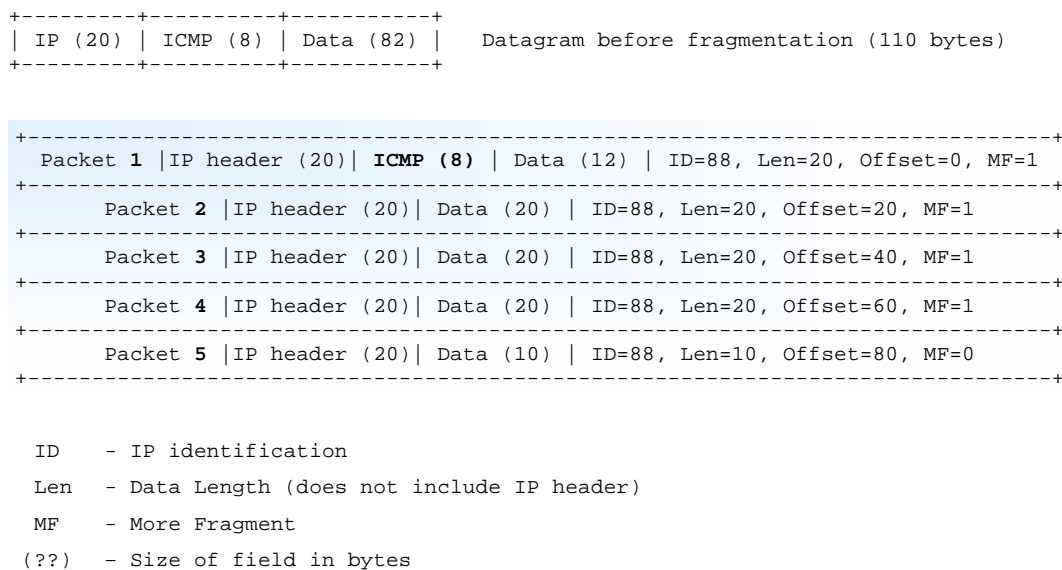
It is clear that the ‘Ping of Death’ is not the result of poor security in the ICMP protocol, but internal flaws in IPv4 fragmentation. Attackers can employ fragmentation overlaps on almost any service using IP datagrams (the fundamental building block of the Internet). The only solution is to secure the system kernel against overflow when reconstructing IP fragments. Almost all patches do this, but attackers can abuse IP fragmentation in other ways. By examining the mechanism in more detail, other exploits become apparent.

7.2.1 Fragmentation example

With IP fragmentation, packets are not reassembled until they reach their final destination. Assembly occurs in the IP layer at the receiving end. This makes fragmentation and reassembly transparent to other transport layer protocols (e.g. TCP, UDP).

Each fragment is associated with an identical fragment identification number, or ‘frag’ ID for short. The frag ID is actually a copy of the IP ID field (IP identification number) in the IP header. Each fragment must carry its ‘position’ or ‘offset’ in the original unfragmented packet. Thus the first fragment will have an offset of 0, and offset counting starts from there. Each fragment must also hold the length of data that it carries. Finally, each fragment must flag the MF (more fragments) bit if it is not the last fragment in the overall datagram.

Consider a 110 byte ICMP packet on a network with an MTU of 40 (very small, for illustration purposes). The datagram would be fragmented into the following sequence of packets.

Figure 7 – IP Fragmentation example

The 110 byte datagram is broken into 5 packets, with total lengths of 40, 40, 40, 40 and 30 bytes. The ICMP data is broken into lengths of 12, 20, 20, 20, and 10 bytes. Notice that the second packet and subsequent packets contain the IP header copied from the *first* packet. There are *no* ICMP headers, except in the first packet. It provides the only data related to protocol identification. This is the case for UDP, TCP and all other protocols using IP in the network layer.

7.2.2 Exploits

In order to exploit IP fragmentation, the attacker must first know the maximum transmission unit (MTU) of the victim's network. This can be found by sending various sizes of ICMP echo requests to the target network with the 'don't fragment' (DF) bit set. If the packet exceeds the MTU, the receiving end will reply with an ICMP error message. The attacker can then lower the size of the packet, and keep sending until no error is returned. There are many 'helper' tools that provide this function.

After discovering the MTU, the attacker can make use of a number of exploits. For example;

- Send fragmented packets to a packet filter that drops inbound ICMP echo requests. The rest of the fragments *will* get in, since protocol information is only contained in the first packet. When receiving subsequent fragments without the initial fragment, the victim will keep trying to perform packet reassembly. This may cause an error, or crash the packet filter entirely.
- Send fragmented packets with identical offset values. The victim will be unable to reassemble the packet correctly. This could be used as a denial of service attack, since the victim machine may hang until valid fragments with the correct offset arrive.

- A well known attack called ‘Teardrop’ ^[21] works by sending packets with spoofed offsets. For example, if the first offset is 0 and data length is 40. The next packet's data length is 10, and offset 20. This is a problem for the victim, in order to reassemble this packet; it must use a negative 20 reference value. Old and un-patched operating systems do not deal well with negative numbers, since they are sometimes translated to very large positive numbers.
- Denial of service on a router is possible by never sending the last fragment – packets are always sent to the victim with the promise of more fragments (i.e. MF bit set). The victim will become overloaded in the hope of receiving more packets for reassembly.

Summary

It is clear programming flaws offer the attacker a large number of exploits. There are hundreds of well known and hidden exploits in even the most established software. IP fragmentation alone, gives the attacker the opportunity to launch at least four different DoS scenarios. Unlike protocol based attacks (e.g. SYN flooding) vendor patches and updates can remove these vulnerabilities. However, with more time spent on the design and testing stages, programmers could eliminate these problems before they become serious threats. Unfortunately tight software release schedules and human error mean programming flaws will always be a problem.

8 Distributed Denial of Service

8.1 Attack concept

Before January 2000, the concept of a distributed denial of service (DDoS) attack was mere theory and rumour. When the first mass attack came in February of that same year, it took down more than six major websites including, Yahoo!, CNN, eBay, Amazon, Buy.com, E*TRADE and countless others [22]. CNN and other victims claim the attack caused damages totalling \$1.7 billion. The media hooked on these attacks and blamed elite teams of hackers for the disruption. In practice the attack could be launched by a single person using freely available, easy to use software. Nowadays almost all major denial of service attacks are distributed.

A DDoS attack is one in which an attacker installs daemons on large number of compromised hosts. At a later point, the attacker sends a request to the daemon asking it to begin flooding a victim with various types of packets. The flood can take any form e.g. a SMURF attack, SYN flood or a constant stream of legitimate packets. The ensuing massive stream of data overwhelms the victim's hosts or routers, rendering them unable to provide a service. Thus DDoS attacks create large scale 'bandwidth consumption' conditions.

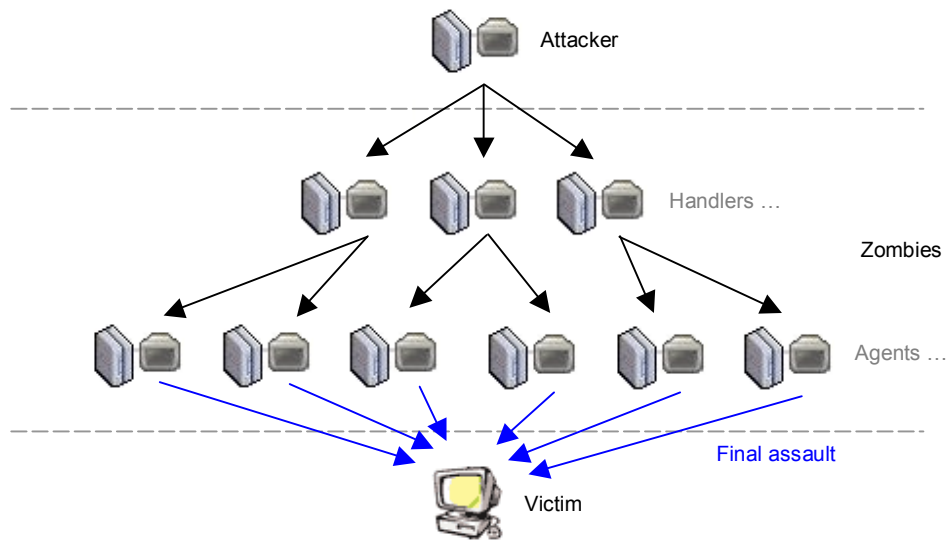
The first step for any DDoS attacker is to target and gain administrative access on as many systems as possible. This task is usually performed with a customised attack script that automatically identifies vulnerable systems. The process can be divided into the following steps, in which the attackers:

- Initiate a scan on a large number of hosts (100,000 or more) probing for vulnerabilities.
- Compromise the vulnerable hosts to gain access.
- Install the DDoS tool on each host (acting as an DoS *Agent*), and/or
- Use the compromised hosts for further scanning and compromising. (act as a *Handler*)

Because an automated process is used, attackers can compromise and install the tool on a single host in under 5 seconds. In other words, several thousand hosts can be compromised in an hour. The hosts or 'zombies', are usually Linux and SUN computers; however, DDoS tools can be ported to other platforms.

After gaining access to a large number of systems, attackers will upload their chosen DDoS program to each zombie. At this point the attacker is poised to launch the attack whenever they wish. The compromised machines have no knowledge they are about to participate in a large scale attack, and the attacker is completely removed from any trace attempt. All flood traffic will be generated from these zombie machines. The figure below illustrates this process, showing multiple-system compromising and the final assault.

Figure 8 – The DDoS attack



Agents have a copy of the chosen DDoS (server) installed. Handlers scan and compromise more systems that will act as Agents.

A *Handler* is a compromised host with a special automated program running on it, scanning for more vulnerable systems. Each handler is also capable of controlling multiple agents. An *Agent* is a compromised host that has the chosen DDoS tool installed. Each agent will be responsible for generating a stream of packets directed toward the victim.

The number of DDoS tools grows almost monthly, a complete analysis of them all is beyond the scope of this project. Some popular, well known DDoS tools will be discussed; Trin00, TFN, TFN2K and Stacheldraht.

8.2 DDoS attacks

Trin00

This attack works by setting up a network of Trin00 clients and servers. A remote controlled program (client) communicates with a master that instructs the daemons (servers) to attack. Communication between client and master is over TCP port 27665 and uses password based authentication. The communication between the master and the server is over UDP on port 27444. Communication from the server back to the master uses UDP on port 31355. Trin00 targets Sun Solaris 2.x and Linux (due to the availability of network sniffers and ‘root kits’ for concealing back doors, etc.)

The attacker controls one or more ‘master’ servers, each of which can control many ‘daemons’ (known in Trin00 code as "Bcast", or "broadcast" hosts.) The daemons are all instructed to coordinate a packet based attack against one or more victim systems. All that is needed is the ability to establish a TCP connection to the master hosts using ‘telnet’, and the password to the master server. The Trin00 network can then be

used to wage a massive, coordinated, denial of service attack. Both the master and daemons are password protected to prevent system administrators (or other hacker groups) from taking control of the network. For a full report on Trin00 and its effects see reference ^[23].

The best way to avoid acting as an agent or handler is to filter the Trin00 port numbers and look for attacking traffic. A number of detection mechanisms exist, including DDoSPing (<http://foundstone.com>) and Zombie Zapper (Razor Team). There is no way to prevent your machine from becoming the victim of a Trin00 network attack.

TFN

The Tribe Flood Network (TFN) attack was written by a hacker named Mixer (one of the most feared DoS authors). TFN has both a client and server component, allowing the attacker to install the server on a remote, compromised system and then with a single command from the client, initiate a full scale DDoS attack. Again, the attack can flood the victim in a variety of ways including, SYN flood, SMURF, ICMP or UDP. TFN can also be configured to gain root access on the victim through a TCP connection.

The TFN attack is more complex than the Trin00 attack because it can execute several attack sequences. TFN uses a variation of the ‘Ping’ function to send commands between the handler and the agents. This makes communication between compromised machines look like legitimate ICMP traffic. To detect a TFN attack, the software must search for an ECHO reply packet that is not preceded by an ECHO request. The same Trin00 detection software (from Foundstone and Razor) can be used against TFN. For a complete technical description of the attack see reference ^[23].

TFN2K

TFN2K or Tribe Flood Network 2000 is the successor to the original TFN attack. It allows the use of randomised communication on ports (meaning port blocking is no longer effective), and Base-64 encryption between hosts (meaning traffic monitoring is no longer effective). Like TFN it can launch SYN, UDP, ICMP floods and the SMURF attack. The TFN2K network can also be configured to randomly switch between these types. This ‘improved’ version is harder to detect and causes more service disruption than its predecessor. For a full explanation see reference ^[23].

Stacheldraht

The Stacheldraht attack (German for ‘barbed wire’) combines the features of Trin00 with those of TFN to provide the most powerful DDoS tool. Similar to TFN, the attack can initiate UDP, SYN and SMURF attacks. Communication between hosts on the Stacheldraht network is achieved by using a combination of TCP and ICMP ECHO reply packets. Packet data is encrypted with a powerful symmetric-key algorithm.

The launching sequence is also password protected. The main difference in a Stacheldraht network is that compromised hosts (agents) can be ‘upgraded’ or ‘patched’ with new versions of the Stacheldraht tool. This process is automated and made possible through *rcp* commands. This allows the DDoS agents to quickly adapt to any new patches or updates that may be installed to remove it. A full description of the attack is available in reference ^[23].

Summary

Distributed denial of service attacks have grown to include the features of many protocol specific DoS attacks. DDoS networks have come a long way from simple protocol exploits, with encrypted communication, random port selection and attack upgrades. They are more complex, effective and harder to trace than standard DoS attacks. Nowadays DDoS is the obvious choice for any attacker. The availability of the tools above, show how easy it is for a novice to launch a denial of service on a massive scale.

The attacks witnessed in February 2000 were a wake up call for many unprepared security experts. Most of these programs were not written by competent experts, but instead cobbled together from bits and pieces of other DoS tools. The next generation of DDoS will be better-engineered, intelligent and more destructive.

9 Conclusions

9.1 Summary

This study has shown that malicious users can launch many types of DoS attacks to disrupt service. Bandwidth consumption attacks allow small amounts of traffic to be amplified to punishing levels. Resource Starvation attacks have been around for many years, and attackers continue to use them with great success. Programming Flaws are a particular favourite of attackers as the complexity of IP stack implementations and Internet software grows. Finally routing and DNS attacks are extremely effective in exploiting vulnerabilities in critical services that make up the core of the Internet.

Distributed denial of service attacks have become increasingly popular due to easy accessibility, and the fact that little programming knowledge is required to launch an attack. These attacks are among the most vicious because they quickly consume all network resources on even the largest hosts, rendering them useless.

Developing the SYN flood program illustrated how easy it is to create a disruptive DoS condition by exploiting a small flaw in the TCP protocol. Writing counter measure software was not so easy, the SYN Detector application can only warn against a SYN flood attack. It is clear protocol designers and indeed *all* network software engineers must consider the reality of DoS attacks. With more time spent on the design and testing stages, programmers could eliminate these problems before they become serious threats.

During the course of this project a destructive Internet ‘worm’ was released with DoS capabilities. So far it has caused large scale disruption for Windows users and gained considerable attention in the media. The ‘MS Blaster’ worm^[24] infects XP and 2000 machines, taking advantage of a known vulnerability in the DCOM (Distributed Component Object Model) interface, which handles messages sent using the RPC (Remote Procedure Call) protocol. The vulnerable systems are connected to the Internet and can be compromised without any interaction from the user.

MS Blaster not only causes problems on the infected PC, but attempts to use the machine as a DoS zombie in a distributed attack against Microsoft’s Windows Update website. The windowsupdate.microsoft.com domain is targeted, preventing Microsoft from simply changing the address of the domain to sidestep the problem. So far the site has withstood the attack and users have been able to download MS Blaster patch software from it. This recent attack proves that DoS is still a favoured tool in the underground community. Since the DDoS tool was so easily available, the attackers simply choose to ‘bolt-on’ this feature to their virus code. It is theorised that all future internet worms will include some form of DoS mechanism.

As e-commerce continues to play a major part in the electronic economy, DoS attacks will have a greater impact on our electronic society. A DoS attack can cause loss of revenue (due to downtime), increased security expenses and lack of customer confidence.

During the recent Gulf war, the Al-Jazeera Iraqi media group experienced a distributed DoS attack on its English and Arabic-language websites ^[3]. The network's websites typically receive traffic in the range of 50 or 60 MB/s. During the attack both sites were hit with traffic in excess of 200 MB/s and up to 300 MB/s. The U.S. based company hosting the sites said it could no longer continue to do so, due to the effect of the attacks on other customer's websites. It is clear that DoS attacks can be used for political reasons or as a propaganda weapon.

9.2 Future Work

With more time and resources it may have been possible to extend this study to include the development and simulation of a distributed denial of service (DDoS) attack. However the lab could not provide enough machines to act as agents in the attack (over 1000). To simulate this realistically would involve illegal activity while compromising a large number of 'zombie' machines.

The SYN Detector application could also be improved to include the following features.

- Use structs or an additional class to create a SYN packet type. To distinguish between SYN packets on individual ports and from different unreachable source addresses.
- Distinguish between a Random SYN flood attack and a standard SYN flood.
- Allow the user to specify the MAXNUMSYNS value in the main dialog window.
- Provide more information on SYN packets including full TCP/IP headers and packet data.
- On detecting a SYN flood, attempt to shutdown or block SYN packets that originate from the unreachable source address.

References

1. Philip Preville, The Montreal Mirror Online Archive, *On the trail of Mafiaboy* (Jun 2000)
<http://www.montrealmirror.com/ARCHIVES/2000/022400/news1.html>
2. Jargon File Resources, *The New Hacker's Dictionary* (Sep 1996)
http://www.jargon.8hz.com/jargon_toc.html
3. Paul Roberts, IDG News Service, *Al-Jazeera Sites Hit With Denial-of-Service Attacks* (Mar 2003)
<http://www.nwfusion.com/news/2003/0326aljahobbl.html>
4. Robert R. Collins, *The Pentium f00f Bug* (Mar 1995)
<http://www.rcollins.org/ddj/May98/F00FBug.html>
5. Purdue University, Christoph Scuba, *Addressing weaknesses in the DNS protocol* (Aug 1993)
<http://ftp.cerias.purdue.edu/pub/papers/christoph-schuba/schuba-DNS-msthesis.pdf>
6. FAQs.org, *Internet RFC/STD/FYI/BCP Archives* (May 2003)
<http://www.faqs.org/rfcs/>
7. Komodia Inc. *TCP/IP library version 4* (© 2003)
<http://www.komodias.com/tools.htm>
8. Tazmen Technologies LLC. *Packetizer version 0.6.6* (© 2002, 2003)
<http://www.packetizer.com>
9. University of California, *WinPcap v3.0* (Jun 2002)
<http://winpcap.polito.it/>
10. Ethereal open source users group, *Ethereal v0.9.14* (Jul 2003)
<http://www.ethereal.com/>
11. Apache Software Foundation, *Apache 2.0.47 http Server* (Apr 2003)
<http://httpd.apache.org/>
12. Microsoft Knowledge Base, Q142641 *Server Unavailable Because of Malicious SYN Attacks* (Jul 2003)
<http://support.microsoft.com/default.aspx?scid=kb;en-us;142641>
13. PowerTech Information Systems, *Smurf Amplifier Registry (SAR)* (Jul 2003)
<http://www.powertech.no/smurf/>
14. US Government, *The National Strategy to Secure Cyberspace* (2003)
<http://www.whitehouse.gov/pcipb>
15. Men & Mice.com, *Single Point of Failure Research* (Jan 2001)
http://www.menandmice.com/6000/6300_single_point_failure.html
16. Robert MacMillan, The Washington Post, *Attack on Internet largest ever* (Oct 2002)
<http://www.washingtonpost.com/wp-dyn/articles/A828-2002Oct22.html>
17. Vagner Sacramento, *BIND DNS Spoofing advisory* (Nov 2002)
<http://www.rnp.br/cais/alertas/2002/cais-ALR-19112002a.html>
18. X5 Networks FAQ, Unknown Author, *The Birthday Paradox* (Unknown)
<http://www.x5.net/faqs/crypto/q95.html>
19. Sans Institute, David Hinshelwood, *DNS, DNSSEC and the Future* (Feb 2003)
<http://www.sans.org/rr/paper.php?id=1054>



-
20. The new FLOWServ, *The Ping o' Death Page* (Jan 1997)
<http://flowserv.teco.edu/ping/>
 21. CERT Advisory, *IP Denial of Service attacks 'Teardrop' and 'Land'* (May 1998)
<http://www.cert.org/advisories/CA-1997-28.html>
 22. CNN.com, *The denial of service aftermath* (Feb 2000)
<http://www.cnn.com/2000/TECH/computing/02/14/dos.aftermath.idg/index.html>
 23. David Dittrich, University of Washington, *Analysis of Distributed DoS tools* (Oct 1999)
<http://staff.washington.edu/dittrich/misc/ddos>
 24. Paul Roberts, IDG News Service, *Windows Blaster worm spreading, experts warn of attack* (Aug 2003)
<http://maccentral.macworld.com/news/2003/08/12/blaster/>

APPENDIX A – Synflooder program source – <http://hutchinson.ijug.org/dos/source/Synflooder.zip>

APPENDIX B – RandSynflooder program source – <http://hutchinson.ijug.org/dos/source/RandSynflooder.zip>

APPENDIX C – SynDectector application source - <http://hutchinson.ijug.org/dos/source/SynDetector.zip>

(For any broken links or other queries contact me at m.hutchinson@ijug.org)

