Evaluating Host-based Methods for Detecting Spoofed IP Packets

An Honors Thesis

Presented to

The Faculty of the Department of Computer Science

Washington and Lee University

In Partial Fulfillment Of the Requirements for

Honors in Computer Science

by

Peter Liudmilov Djalaliev

2005

To my mother Donka, brother Milin and sister Gaby

Contents

ACKNOWLEDGMENTS

First, I would like to thank Professor Rance Necaise for all his guidance and advice throughout this project. I would also like to thank Proferssor Whaley, Professor Levy and Professor Lambert from the Computer Science department for all their help during my four years at Washington and Lee. Last but not least, I owe my deepest gratitude to my family and friends for their constant support wherever my endeavors decide to take me.

ABSTRACT

Billions of messages are transmitted across the Internet each day and most of these use the Internet Protocol (IP) to route packets to specific destinations based upon an IP address. While these packets contain both source and destination addresses, the protocol provides no means to verify the authenticity of the source. Therefore, packets can be sent with intentionally altered source addresses, known as spoofing, which in most cases is done for malicious purposes. IP spoofing is an integral part of various distributed denial-of-service attacks. Today, a number of methods of detecting spoofed IP packets have been studied in order to limit possible damage. These detection methods are classified as router-based and host-based, depending upon the site of implementation. Host-based detection methods are of particular interest because they can be implemented locally irregardless of the Internet service providers. The current research on host-based methods provides a number of possible solutions. However, there is insufficient data related to their efficacy. We evaluate the performance of the well known host-based spoofing detection methods under various circumstances and explore the ability of these methods to complement one another in order to improve their efficiency.

Chapter 1

Introduction

Billions of messages are transmitted across the Internet every day. While most of them are legitimate, some are transmitted with bad intentions. The most common mechanism for transmitting data over the Internet is with the use of the Internet Protocol (IP).

A valid IP message requires the correct address of both the originating and destination hosts. There is no mechanism for verifying these addresses. If a message contains an invalid destination address, it is simply not delivered. An invalid or fake source address, however, allows for the transmission of fake and possible destructive messages.

IP spoofing is the name given to the transmission of data over the Internet with fake source addresses. This technique is used in a number of elaborate network attacks. IP spoofing is used either to hide the real identity of the attacker and the source of the attack, or to cause an Internet host to act as a reflector – replying with multiple messages to pre-selected victims.

The countermeasures proposed by research to defend from spoofing attacks are detection and backtracking. Detecting spoofed packets can help blocking attacks before they reach

Chapter 1

Introduction

Billions of messages are transmitted across the Internet every day. While most of them are legitimate, some are transmitted with bad intentions. The most common mechanism for transmitting data over the Internet is with the use of the Internet Protocol (IP).

A valid IP message requires the correct address of both the originating and destination hosts. There is no mechanism for verifying these addresses. If a message contains an invalid destination address, it is simply not delivered. An invalid or fake source address, however, allows for the transmission of fake and possible destructive messages.

IP spoofing is the name given to the transmission of data over the Internet with fake source addresses. This technique is used in a number of elaborate network attacks. IP spoofing is used either to hide the real identity of the attacker and the source of the attack, or to cause an Internet host to act as a reflector – replying with multiple messages to pre-selected victims.

The countermeasures proposed by research to defend from spoofing attacks are detection and backtracking. Detecting spoofed packets can help blocking attacks before they reach the victim host. Backtracking the detected spoofed packets will allow discovering the real source of the attack. Spoofed IP packet detection methods can be implemented either on the routers between the attacker and the victim (router-based) or on the local network of the victim host (host-based). Host-based methods have the advantage of being immediately deployable in the case of an attack. Even though such methods have been proposed and studied by previous research, little data is available about their performance.

For this research, we implemented and evaluated the performance of three host-based spoofed IP detection methods using a virtual network simulation package. Combining multiple detection methods, our goal was to minimize the number of false positives and false negatives returned by the detection process. The results suggest that we need to implement more methods in order to build a dependable and robust detection system.

Chapter 2

Background

2.1 Network Basics

A *computer network* is a system for communications among two or more computers [23]. Today, computer networks are built from general-purpose programmable hardware devices and can be used to transmit various types of data [17]. A network generally consists of various hardware components.

A computer involved in communications with one or more other computers on the network is called a *host.* The intermediate nodes, which assist in communications are called *network devices* (i.e. network switches and routers).

The communication between computers on a network is performed with the transmission of messages between those computers. In order to transmit messages, the two computers must agree on a specific data format and rules of communication. This data format and rule specification is known as a protocol. Protocols are usually implemented inside *protocol stacks*, which have a layered structure. The protocols in each layer are specialized in handling a specific portion of the data transfer process between two hosts. A protocol usually provides services only inside a single layer, thus allowing protocols to specialize, as well as helping make clearer distinctions between the scopes of functionality to be provided by the different protocols.

A protocol at one layer typically uses a protocol in the layer below to transmit messages. On the receipt of a message, a protocol hands the message or a subset of it off to a protocol from the layer above. Each protocol specifies two different interfaces. It provides a *service interface* to the protocols and applications services from the layers above. Also, it specifies a *peer interface*, which is used by the identical protocol on the other side of the communication link to send or receive data to its peer [17]. Using interfaces achieves both encapsulation and specialization of services. Each protocol encapsulates its specific internal functionality hidden from the outside layer, as long as it provides the required service and peer interfaces.

2.2 The TCP/IP Protocol Stack

The set of protocols that has been accepted as the de facto standard for communication over the Internet is the TCP/IP protocol stack, named after its two most important members. The protocol stack consists of multiple protocols, each of them providing a specific service, which include routing and packet fragmentation, reliable connections and message acknowledgment, remote file transfer, e-mail messaging and transfer of hypertext used in websites.

The TCP/IP protocol stack consists of five layers, each of which contains multiple protocols or specifications [23]. Messages are transmitted down the protocol stack from the application of the source host to the transport layer and then down to the physical connection, as illustrated by Figure 2.1. Messages are received by the destination host at the physical connection. The messages then travel up the protocol stack to the application layer.

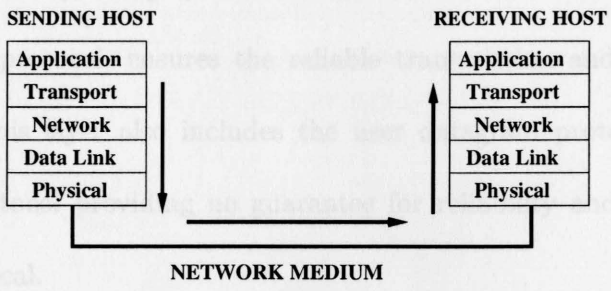

Figure 2.1: The movement of a message on the TCP/IP protocol stack.

The *physical layer* handles the physical transmission of data. This layer is responsible for the specifications of different transfer media (coaxial, twisted-pair and fiber optic cables; radio and infrared waves), types of connectors (RJ-45 connectors used in Ethernet cables), signal strength and modulation.

The *data link layer* deals with the transmission of data in a local area network confined to a limited area, such as a college campus. It specifies how data is formatted when transferred over the physical media. This layer contains multiple protocols which specify the frame format (a single unit of data transferred over the physical media), as well as the mechanism used by multiple devices to access shared transfer media. Examples of such protocols are Ethernet, SLIP (used by modems for dial-up Internet connections), the IBM Token Ring and ATM (Asynchronous Transfer Mechanism). Ethernet is one of the most popular data link layer protocols used today.

The third layer is the *network layer*, which handles routing of packets, the single units of data at this layer, in a wide area network. This protocol layer supports packet fragmentation (IP) when required by the underlying physical network, as well as intra- and inter-domain routing (BGP, OSPF, RIP).

Next comes the *transport layer*, in which the transmission control protocol (TCP), a connection-oriented protocol, ensures the reliable transmission and receipt of messages, called segments. This layer also includes the user datagram protocol (UDP), which is a connectionless protocol providing no guarantee for reliability and used in cases where reliability is not critical.

At the top of the protocol stack is the *application layer*, which provides support for network applications. This layer contains protocols, used by applications to provide services such as e-mail messaging (SMTP, POP, IMAP), remote file transfer (FTP) and hypertext transfer (HTTP).

When transferring a message, a host needs to be able to uniquely identify the source and the destination of the transfer. Because such identification is required at multiple stages during the process of sending or receiving a message, each of the Data Link, Network and Transport layers of the TCP/IP protocol stack applies a certain addressing scheme.

2.2.1 Internet Protocol

The Internet Protocol (IP) is defined at the network layer of the Internet protocol stack. It handles IP addressing and routing, as well as packet fragmentation where needed. As Figure 2.2 shows, the packet consists of a 20-byte header and packet data. The header contains two 32-bit IP addresses, which uniquely identify the target and the destination of the packet [20].

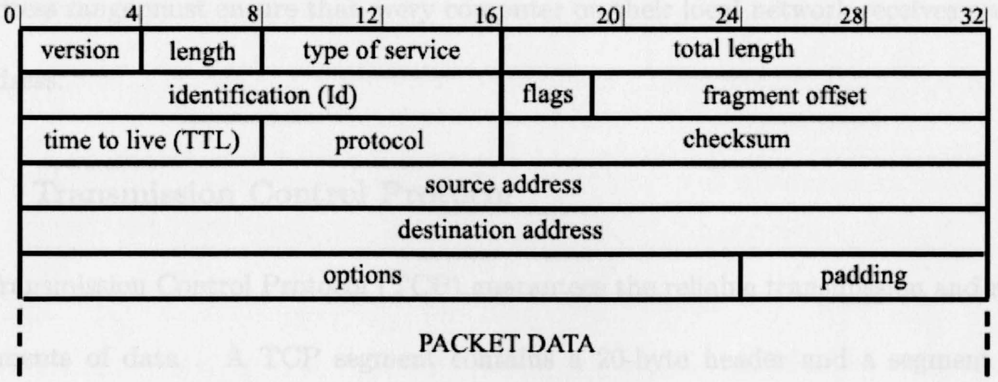

Figure 2.2: The structure of an IP packet.

IP addresses are identified using four 8-bit integers, ranging from 0 to 255 ($2^8 - 1$). Every IP address is divided into two parts - network and host. The network portion of the IP address identifies the local IP subnet and is shared by all hosts on that subnet. The host portion identifies a host on that IP subnet and must be unique for all hosts. The exact number of bytes allocated for the network and host portions depend on the size of the networks and the number of hosts in it. For example, an IP address with an 8-bit network portion and a 24-bit host portion provides 255 ($2^8 - 1$) possible network identifiers and 16777215 ($2^{24} - 1$) possible host identifiers in each network. On the contrary, an IP address with a 24-bit network and 8-bit host portions allows for 16777215 distinct network identifiers and only 255 possible host identifiers within a network.

Each network adapter must have a unique IP address no matter where it is located in the world. The uniqueness of IP addresses is guaranteed by organizations, such as the Internet Assigned Numbers Authority (IANA), which assign ranges of IP addresses to any institution that requests them. Once an institution has been assigned an IP range, it can assign portions of that range to other companies and institutions. The final consumer of an IP address range must ensure that every computer on their local network receives a unique IP address.

2.2.2 Transmission Control Protocol

The Transmission Control Protocol (TCP) guarantees the reliable transmission and receipt of segments of data . A TCP segment contains a 20-byte header and a segment body. The size of the body is limited in most cases to the maximum frame size of the underlying network.

The TCP header, as shown in Figure 2.3, contains two 16-bit port numbers, which identify the source and destination of the TCP segment. These port numbers, together with the source and destination IP addresses, can uniquely identify a TCP connection between two hosts on the network. Port numbers are numeric labels for the operating system ports – abstract points of entry for remote network services into the system.

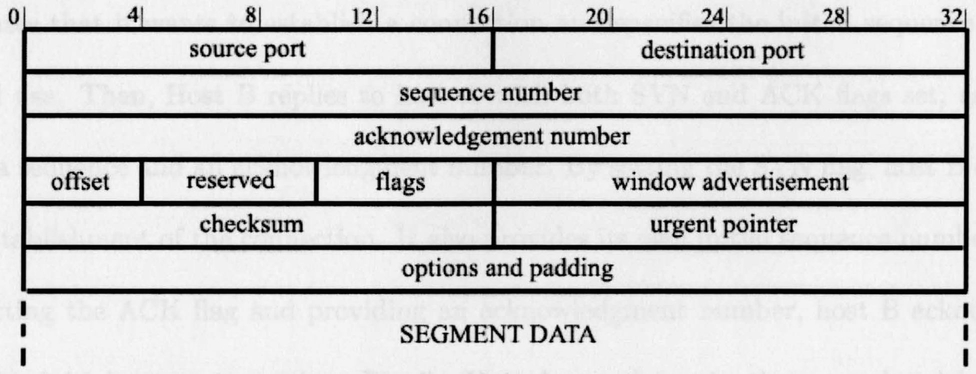

Figure 2.3: The structure of a TCP segment.

The 32-bit sequence and acknowledgment numbers are a part of the protocol automatic acknowledgment mechanism. Every segment sent along a TCP connection from host A to host B has a unique for that connection at that time sequence number, which identifies the segment among the others transferred along the same connection. After B receives a segment from A, it sends a reply with its acknowledgment number set to the sequence number of the original segment. In this way, host A can confirm that every segment it sends to host B has been received.

The TCP header also has 6 flags, which are used to exchange control information during communication. The four most important ones are Synchronize (SYN – used to establish and synchronize a connection), Finalize (FIN – when terminating a connection), Acknowledge (ACK – any time when the segment is acknowledging the receipt of another segment) and Reset (RST – when one side wishes to abort the connection).

The first step in any TCP connection is the initial three-way handshake. It serves to confirm the real identity of both hosts and to establish the initial sequence numbers that they are going to use during communication. As shown in Figure 2.4, Host A first sends a segment to host B with a sequence number and the SYN flag turned on. With this, host A indicates that it wants to establish a connection and specifies the initial sequence number it will use. Then, Host B replies to host A with both SYN and ACK flags set, as well as both a sequence and an acknowledgment number. By setting the SYN flag, host B confirms the establishment of the connection. It also provides its own initial sequence number. Also, by setting the ACK flag and providing an acknowledgment number, host B acknowledges host A's initial sequence number. Finally, Host A completes the three-way handshake with a segment, which has its ACK flag turned on, as well as an acknowledgment number, with which it acknowledges host B's initial sequence number [17].

If the handshake is successful, the connection is established and the two hosts can

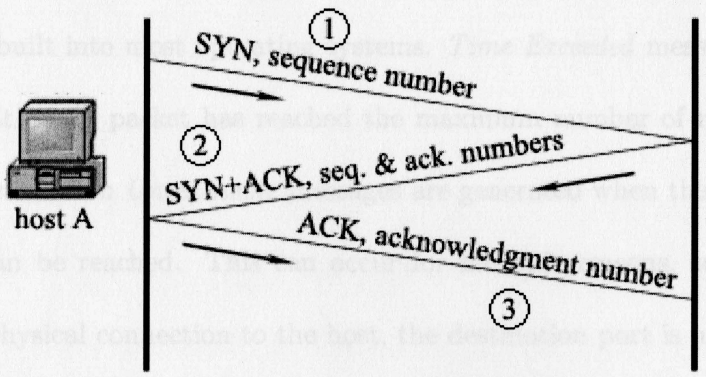

Figure 2.4: The three-way initial TCP handshake.

start exchanging data. If at any point an unexpected event occurs, a host will abort the connection by replying with a segment that has its RST flag set. This can occur if the host receives data over TCP from a host with which it has not established connection, or it receives an acknowledgment for a sequence number it has not sent.

2.2.3 Internet Control Message Protocol

The Internet Control Message Protocol (ICMP) is a part of the Internet protocol suite, which operates on the Network level and is mainly used by the IP protocol when an IP packet requires a response to the sender. When such a case occurs, an ICMP message is generated and placed inside an IP packet, which is then passed down to the data link layer. Thus, ICMP operates as if it was a higher-level protocol, but in fact it is an integral part of the IP protocol.

Every ICMP control message has a type, which specifies the reason, for which it was generated. Among the more important control message types are *Echo Request*, *Echo Reply*, *Destination Unreachable* and *Time Exceeded*.

Echo Request and *Echo Reply* are the most common ones because they are used by the *ping* utility built into most operating systems. *Time Exceeded* messages are used to notify a sender that an IP packet has reached the maximum number of router hops before it is dropped. *Destination Unreachable* messages are generated when the destination host of an IP packet can be reached. This can occur for multiple reasons, some of which are that there is no physical connection to the host, the destination port is not accessible, or that a packet, which the sender has forbidden to be fragmented, exceeds the maximum size limit for a single message of a physical network link along the route [18].

2.3 Internet Address Spoofing

Messages transmitted across the Internet contain source and destination IP addresses used in routing the messages between the interconnected network components. When a packet leaves the source host, the current IP protocol has no built-in mechanism for verifying the source IP address as the packet travels from one router to another. Most operating systems restrict user access only to protocols at the Application layer. However, some systems, such as UNIX, provide the ability to bypass levels of the protocol stack and directly access the Data Link layer [12]. This allows malicious users to send custom-built IP packets with altered, or spoofed, source IP addresses to any host on the Internet without being detected.

IP spoofing is used primarily for malicious purposes and is an integral part of most significant network attacks. The two main purposes of spoofing addresses by malicious users is to hide their identity and to make an attack appear to come from a large number of random hosts. This makes it harder for the firewall or intrusion detection system to filter out these packets.

IP spoofing can also be used to mislead remote hosts into acting as *reflectors* and forwarding large amounts of traffic to the victim host. Some servers are potential reflectors because they will automatically reply to certain types of TCP segments, such as HTTP requests and DNS queries. Other messages invoking automatic replies are ICMP Time Exceeded and Host Unreachable messages [21]. If the attacker sends such messages to a sufficiently large number of reflectors with the source IP address spoofed to the address of the victim, they will then flood the victim with automated replies. This can result in a denial of service attack.

2.4 Network Attacks Involving IP Spoofing

Spoofed IP traffic is used in many types of attacks over the Internet today, both in order to hide the attacker's real identity and to mislead remote hosts into acting as reflectors.

2.4.1 SYN-Flooding

Synchronize (SYN) flooding exploits the three-way handshake feature of the TCP protocol [21]. If an attacker sends an initial SYN segment requesting a connection, the victim host will return a SYN/ACK segment with an acknowledgment and its own sequence number. The victim host will reserve buffer space in its memory for the system function waiting for the final acknowledgment. If the final acknowledgment is not received in a specified amount of time, the connection will time out and the buffer space will be freed.

This handshaking sequence can be exploited as illustrated in Figure 2.5. The attacker can generate a sufficiently large number of connection requests to deplete the victim's buffer space for new connections, thus forcing it to deny connection requests from legitimate sources.

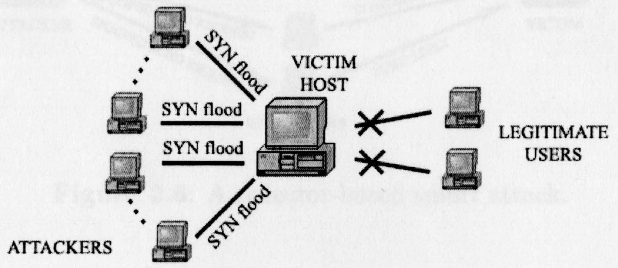

Figure 2.5: A SYN-flooding attack.

In this attack, the host, on which the malicious user is working, does not require the acknowledgment returned from the attacked host. This allows the source addresses of the IP packets to be safely spoofed inside the outgoing segments without detection. The spoofed address, however, must not belong to a valid host. Otherwise, the victim will send back the acknowledgment to the valid address, receive a segment with the RST instead of the ACK flag set and will release the reserved buffer space, which will prevent a buffer overflow.

2.4.2 Smurf Attacks

An attacker can use reflector hosts in a smurf attack, in which the victim sends a large number of ICMP Echo requests to reflector hosts with the source address spoofed to that of the victim. This results in the reflectors flooding the victim's network with a large number of Echo replies. This malicious traffic consumes both the bandwidth of the victim's local network, as well as some of its CPU time because it needs to process each incoming Echo reply.

The use of reflectors makes the attack more powerful. Firewalls are more likely to filter

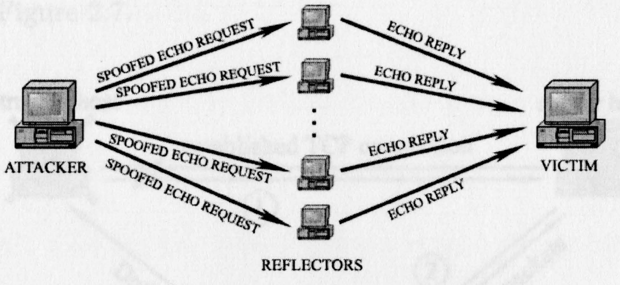

Figure 2.6: A reflector-based smurf attack.

Echo requests than Echo replies [21]. Reflectors can often act as packet amplifiers, sending larger-sized replies than the requests they receive from the attacker, which increases the load on the victim's network bandwidth. Reflectors also make filtering and traceback of the attack harder. It takes considerably more time for a host on the victim's network to monitor traffic coming from a large number of reflectors. While the victim could possibly trace the attack back to the reflectors, all packets sent from the attacker to the reflectors were spoofed, so tracing the attack back to the attacker would be very difficult.

2.4.3 TCP Connection Hijacking

When a remote client host successfully establishes a TCP connection with a server host, it is considered a trusted host for that server. If an attacker can hijack that TCP connection and disguise himself as the trusted host in front of that server, then it would receive the same access privileges as the trusted host itself.

TCP connection hijacking involves the coordination of multiple attacking techniques. The attacker sends IP packets to the victim with source addresses spoofed to the address of the trusted host. Also, the attacker performs a denial-of-service attack to the trusted host, so it cannot respond to any packets coming from the victim server. The steps of the attack can be observed in Figure 2.7.

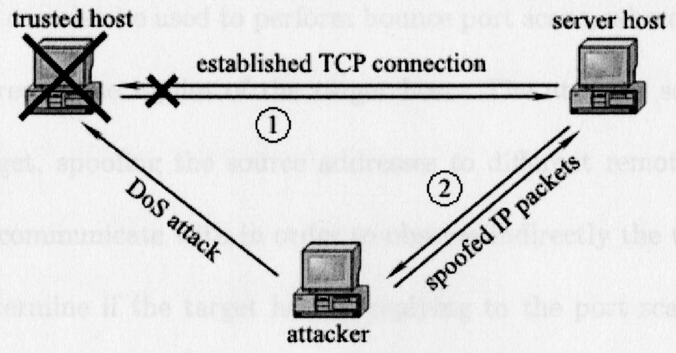

Figure 2.7: The steps of a TCP connection hijacking attack.

In the beginning, the remote host has established a TCP connection with the server, thus becoming a trusted host of that server. The attacker carries out a denial of service attack on the trusted host, preventing it from replying to any packets coming from the server. The attacker initiates communication with the server host, spoofing its outgoing IP packets to the IP address of the trusted host. For a certain period of time, the server does not realize that it is communicating with a different entity on the network, so the attacker gains trusted host access privileges to that server.

In order for the hijacking to remain unnoticed by the server host, the attacker needs to guess the next sequence number expected by the server host. Otherwise, the server will reset the TCP connection and the attack will fail. RFC 1948 provides recommendations on how to make TCP sequence numbers difficult to guess, but in practice they are still guessable [3].

2.4.4 Bounce Port Scanning

IP packet spoofing can also be used to perform bounce port scans, where the malicious user observes only indirectly the replies of the target host. The attacker sends port scanning packets to the target, spoofing the source addresses to different remote hosts, which the attacker can later communicate with in order to observe indirectly the target's replies.

In order to determine if the target host is replying to the port scanning messages, it sends IP packets to the spoofed remote hosts and observes their IP identification field. If the targeted port is open, it will reply to the spoofed host, thus causing it to increment its IP Id field. If the port is closed, the victim will send back to the spoofed host a segment with the RST flag set, which performs no action upon its receipt [21]. Thus, the malicious user can send an IP packet to the soon-to-be spoofed host beforehand. It then performs the scanning and finally sends another IP packet to the spoofed host and observes if its IP Id value has been incremented beforehand. In order for this scenario to work, the spoofed hosts must have no other outgoing IP traffic. Otherwise, any outgoing IP packets would cause the the IP Id value of the spoofed hosts to increment without receiving a reply from the victim host. Therefore, the port scanning technique will report some closed ports as open.

2.4.5 Zombie Control

In distributed denial-of-service attacks, the malicious user often makes use of zombie hosts, which perform the real attack on its command [21]. Using zombies increases the intensity of the Distributed Denial Of Service attack because the flooding traffic is generated from a greater number of sources. Virtually any host on the network which the attacker can compromise, can serve as a zombie. An agent on a zombie host opens a listening socket and waits for commands from the attacker. Generally, the attacker does not require any replies from the zombie, so it can safely spoof the source IP addresses of the attacking commands, thus hiding the real source of the attack.

2.5 Defense from IP Spoofing Attacks

Counteracting IP spoofing would be an integral step towards defending from the attacks described in the previous sections. Existing research on IP spoofing is concentrated on detection and backtracking as methods of counteraction. Spoofed IP traffic detection techniques are divided into host-based and router-based. The former operates at the site of the host under a spoofing attack, while the latter attempts to detect spoofed packets on their way to the victim. Being able to detect spoofed packets would make spoofing attacks much harder to execute.

There have been attempts to develop mechanisms for backtracking spoofed IP packets, which involves tracing the route of the spoofed packets back to the their real source. This is an important step for identifying attacking hosts, which use spoofed traffic to hide their real identity.

Dunigan [8] has proposed multiple automated ways to backtrack spoofed IP traffic from the destination back to its real source host. As he notes, the manual logging into each upstream router and checking the interface, on which a packet was received, is a very slow process. Also, this is sometimes even impossible because it would require physical access to the router. Therefore, he has proposed a number of tools to automate this process.

For example, DosTracker [8], which is a Perl-based script written for Cisco routers, can log into each upstream router one after another, tracing the path back to the real source host. Another example are tracer daemons [8], which require a tracking agent to be configured on every subnet along the path of the backtracked packet. The target host of its intrusion detection system initiates a backtracking process, which logs into the tracking agent on every subnet moving towards the source of the suspicious packet and checks what subnet the packet came from. A third example is the DECIDUOUS technology [8], which uses the IP Security (IPSec) protocol set's security associations and authenticated header protection. It is based on the assumption that an IP packet must have gone through every router, which has authenticated that packet. By building security associations further and further away from itself, the victim host can trace the spoofed packet to the router closest to its real source host.

The disadvantages of these techniques are that they are difficult to deploy universally. For example, while DosTracker is based on a single router device technology, tracer daemons require the configuration of specialized host on every IP subnet.

2.6 Router-based detection methods

Router-based spoofed packet detection methods are implemented on the routers, through which spoofed packets are expected to pass. The main advantage of such detection methods is that if they are implemented on a sufficiently large portion of the Internet routers, they will help stop spoofing attacks even before they reach the victim host. The main disadvantage is that widespread implementation depends on the policies of multiple router manufacturing companies and Internet service providers. Driven by commercial goals, these companies often lack incentives to implement spoofed packet detection methods, which would provide no direct returns.

2.6.1 Ingress Filtering

RFC 2827 [10] specifies ingress filtering, which is a mechanism for filtering spoofed IP packets at the source's local network point-to-connection with the Internet. Each local network usually coincides with a single IP range assigned to the hosts of that network. If the router at the local network's Internet connection detects a source IP address of a packet outside the expected range, it should drop it. If applied, ingress filtering can prevent most IP spoofing or restrict spoofing to IP addresses within the local IP subnet, which would make discovering the real identity of the attacking host easier. However, application of network ingress filtering has been sporadic again because vendors and ISP's do not have sufficient incentives to adopt the mechanism as a standard.

2.6.2 Pushback

The mechanism of pushback, proposed by Mahajan et al. [14, 11], treats distributed denial-of-service attacks as a congestion control problem. Aggregate-based congestion control (ACC) groups incoming traffic in aggregates according to a common characteristic: protocol type, source IP subnet, etc. According to the ACC pushback mechanism, routers on the way to the victim can request upstream routers to limit the bandwidth allowed for packets from a given aggregate. If a router can identify the signature (the identifying common characteristics) of the traffic flooding the victim, it can pass messages to the upstream routers to limit the bandwidth of that traffic. However, the attacker can use multiple protocols and random IP addresses in order to make the discovery of a clear attack signature almost impossible.

2.6.3 Traceback

Savage et al. [19] proposes a mechanism for IP traceback where every packet carries partial information for the path it has traversed. If a flooded victim receives a sufficient quantity of spoofed packets, it should be able to discover the true source based on the path information in these packets. The IP traceback mechanism has been further developed by Song and Perrig in order to deal with multiple attackers [16].

Bellovin et al. [4] have proposed the ICMP traceback technique. Here, routers on the path traveled by a packet can probabilistically send ICMP messages to the destination host. The destination host logs these ICMP messages and if it receives enough spoofed traffic, it should be able to identify the real source of the spoofed packets.

2.6.4 Overlay Networks

Overlay proxy networks [22, 13, 1] have also been proposed as a way to protect a server from distributed denial of service attacks by hiding the exact location of the server inside a network. Most servers on the Internet are vulnerable to distributed denial of service attacks because they are publicly available. However, if an application is hidden inside a proxy network, with which clients have to communicate without knowing the real destination IP address, this would provide sufficient time to reconfigure the application and move it to another location, thus avoiding the attack.

2.7 Host-Based Detection Methods

Numerous host-based detection techniques have been proposed, which utilize various idiosyncrasies of the TCP/IP protocol stack. Host-based detection methods are divided into passive and active methods according to the amount of resources required to scan an incoming packet.

Passive detection methods simply observe incoming packets and make decisions solely based on information from these packets. Due to low time and memory resource usage required to scan a single packet, passive detection methods can scan all or a large portion of all incoming packets. However, they are not as accurate. Active detection methods require the detection system to send an active probe to a remote host and wait for a reply, thus dedicating more resources in terms of waiting time and a memory buffer for the waiting process. In return, because they perform a more active role in marking a packet as spoofed or legitimate, they are more accurate.

The accuracy of both passive and active methods is measured in terms of the number of *false positives* (legitimate packets marked as spoofed) or *false negatives* (spoofed packets marked as legitimate) generated during the detection process. A large number of false positives implies that the detection system is blocking legitimate packets and, thus, denying service to legitimate clients. False negatives indicate that the detection system is not filtering out all spoofed incoming packets. An effective detection system would be able to minimize both variables.

2.7.1 Passive Detection Methods

Hop-count filtering [5] is a passive detection method, which uses the Time-To-Live (TTL) value of the IP header to mark possibly spoofed detect packets. The detection system stores actual hop counts to remote IP subnets and then compares them to the hop counts of incoming packets. If the two values are different, the packet is marked as possibly spoofed as indicated in Figure 2.8. Hop-count filtering requires the ability to securely update the IP/hop count table when a change in routing paths is detected.

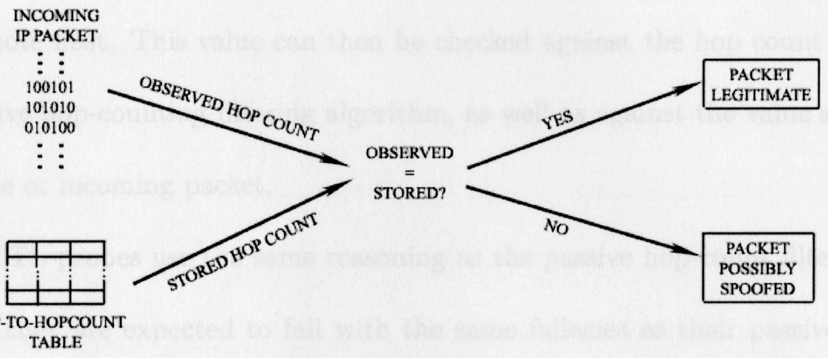

Figure 2.8: Step-by-step execution of the passive hop-count filtering algorithm.

This filtering technique is made possible by different observations about TCP/IP stack implementations and Internet routing paths. The hop count of a packet is measured as the difference between the final and the inferred initial TTL values of the IP header. This measurement is expected to be accurate due to the relatively few, located far apart from each other, initial TTL values used by various TCP/IP implementations [9]. This and the observation that most hosts on the Internet are less than 30 router hops apart from each other [6, 7] make it possible to predict initial TTL values. For example, it can be assumed that the initial TTL values used by most modern operating systems are 30, 32, 60, 64, 128 and 255. If the system receives a packet with a TTL value of 115, then it can safely infer an initial TTL value of 128 and a hop count of $128 - 115 = 13$.

2.7.2 Active Detection Methods

Direct TTL probes are packets sent from the detection system to the remote host, which require a reply [21]. Such packets can be TCP SYN segments (the first packet sent during the initial three-way handshake) or ICMP echo requests, for example. The probe replies received by the detection system can be used to infer the number of router hops it takes to reach a remote host. This value can then be checked against the hop count value inferred by the passive hop-counting filtering algorithm, as well as against the value stored from an earlier probe or incoming packet.

Direct TTL probes use the same reasoning as the passive hop-count filtering detection method, so they are expected to fail with the same fallacies as their passive counterpart. Nevertheless, direct TTL probes provide a more current hop count value, which can be stored for future references. The goal is to check if and when direct TTL probes can provide better detection accuracy compared to the passive hop-count filtering algorithm.

If the direct probe and the original packet are of the same protocol, then we do not need to infer the hop count. Instead, we can just compare the final TTL values because packets of the same protocol usually have the same initial TTL value.

Another way to check packets, marked as spoofed or legitimate by the passive detection algorithms, is to use the IP header identification field, originally used to identify fragments of the same IP packet [20]. When used in combination with the source and destination IP addresses and a given protocol, it uniquely identifies a packet (or any fragment of it) on the

Internet at a single point of time.

The importance of the IP Id value requires the operating system to have an algorithm for managing the pool of 2^{16} possible Id values, so no two packets on the Internet with the same source and protocol can have the same Id value. Most systems accomplish this by using fixed incremental values for each outgoing packet. While older Microsoft-based systems (Windows 9x and NT 4.0) use an incremental value of more than 256, recent versions of Windows (ME and later), as well as most other operating systems, increment the Id field by 1 [2].

With this observation, it can be assumed that two packets leaving the same host in close time proximity have similar identification values. If the detection system sends an IP Id probe immediately after the original packet has been received, the IP Id value of the probe reply should be larger, but close to the Id value of the packet being checked. On the contrary, if the probe Id value is smaller or significantly larger than the one being checked, this could indicate a spoofed packet.

This method, however, would fail for packets coming from systems using a high incremental value, as well as systems applying alternative ways for handling Id value assignment, such as the use of pseudo-random number generators. In this case, each successive Id value depends only on the seed of the random-number generator, so the previous assumptions related to packet Id values in close proximity would not be valid.

Other host-based detection methods proposed by researchers include the use of OS fingerprinting, which involves deducing the operating system running on a remote host by the peculiarity of its protocol stack implementation, the TCP protocol, the *traceroute* program and various OS idiosyncrasies. OS fingerprinting can be used together with direct

TTL probes to identify the operating system used on a remote host. The TCP protocol flow control and packet retransmission algorithms can be used to ensure that certain traffic is indeed coming from the supposed source IP address. Traceroute can provide the actual number of router hops for paths, along which ICMP Echo requests and replies are not blocked by firewalls [21].

Chapter 3

Experiments

Even though multiple detection methods have been proposed in the literature, little data is available related to their performance under different circumstances. Jin et al. [5] evaluates the performance of hop-count filtering, concluding that it can be used to detect 90% of the spoofed traffic from a source. Templeton and Levitt [21] confirm the high-predictability of TTL initial values for ICMP, IGMP, TCP and UDP messages, thus making possible the inference of hop counts needed to reach a remote host, an important part of the successful hop-count filtering process. No research data, however, is available on the individual performance of the other proposed detection methods. Since these methods use different inherent reasoning to make decisions about the validity of packets, we expect them to complement each other's strengths and have better combined detection accuracy in a greater variety of situations. However, no data is available on the combined performance of host-based detection methods, either. The goal of this research is to examine the strengths and weaknesses of three detection methods – passive hop-count filtering, active TTL and IP Id probes.

The performance of the three spoofed IP packet detection algorithms was evaluated by implementing the algorithms in a detection system using the NetSim virtual simulation package [15]. We designed appropriate virtual network topologies and tested the ability of the detection methods to minimize false positives and false negatives. The NetSim virtual network simulation package is described in detail in the Appendix.

3.1 The Detection System

The main detection algorithm in the system is the passive hop-count filtering [5], which filters all incoming packets at the IP protocol level of the NetSim host module. The algorithm infers the number of hops taken, based on the TTL value of the incoming packet, and compares that value to the hop count stored for the same remote IP subnet from a previous point of time. If the two hop counts match, the packet is marked as legitimate; if they differ, it is marked as spoofed.

In order to remember hop count values for legitimate packets, the hop-count filtering algorithm uses an IP-to-hopcount table, which stores observed hop counts for remote IP subnets. For example, if the detection system receives a legitimate packet from a given subnet, it will record the network portion of the IP address of that subnet, the hop count observed and the time when the table entry is created.

Passive hop-count filtering has been expanded further by Jin et al. [5] who propose IP address aggregation to solve the problem of co-located IP subnets. Co-location occurs when the addresses from a single block of the IP address range are assigned to hosts from different physical networks, which possibly lie at different hop counts from the detection system. Even though the hop-count algorithm is a good first line of defense for the detection system, we found that in certain conditions it underperforms in terms of both false negative and false positive results.

3.2 False Negative Experiments

False negatives are a result of the detection system using the final TTL value in order to infer the initial TTL. The algorithm assumes that the most common initial TTL values are $30, 32, 60, 64$ and 128. Using the final TTL field, the algorithm infers the real initial TTL value to be the next largest common initial TTL value above the final value. For example, if the detection system observes a final TTL value of 110, it infers the real initial TTL to have been 128 – the next largest common one above 110. This reasoning is made plausible by the observation that most hosts on the Internet are 30 or less hops away. Thus, a packet with a final TTL value of 110 would not have an initial TTL of 255, which would imply a hop count of 145. As long as the observed final TTL value is located between common initial TTL values, which are far apart from each other, the detection system will in most cases infer the initial TTL field correctly.

This inference, however, will not work when the final TTL value is located just below two tightly-packed common initial values – 30 and 32, or 60 and 64. Let's suppose that the detection system receives an incoming packet with TTL field of 20. Both 30 and 32 would be possible initial TTL values. In such a case with more than one possible initial TTL value, the detection system considers both values.

3.2.1 Passive Hop-Count Filtering and False Negatives

Figure 3.1 illustrates a network topology with a chain of 10 routers. On one end of that chain is subnet 1 where the detection system host is located. Subnets 2 and 3, at the other end of the chain, are respectively 6 and 10 hops away from subnet 1. Subnet 3 contains multiple hosts using 64 as the initial TTL value. This is always possible because 64 is a common initial TTL value and we can pick subnet 3 to be any subnet 10 hops away from the detection system.

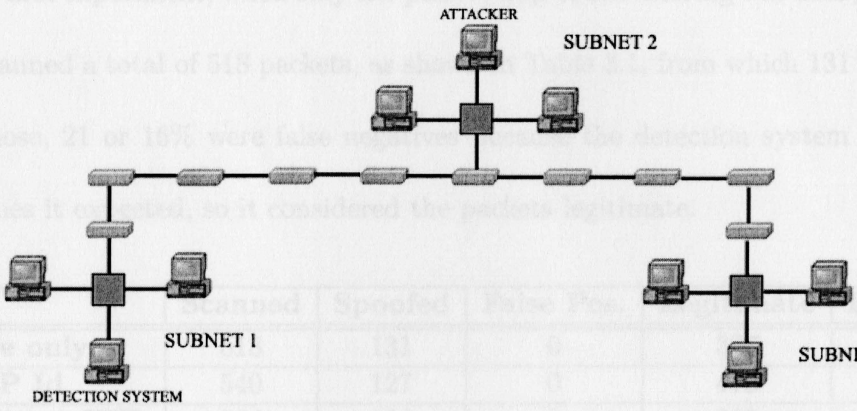

Figure 3.1: A sample network topology.

During normal IP traffic, packets sent from subnet 2 and subnet 3 with initial values of 64 will arrive at subnet 1 with final TTL values of 58 and 54, respectively. In both cases, the possible inferred initial TTL values would be 60 and 64, resulting in possible hop counts of 2 $(60-58)$ and 6 $(64-58)$ for the packet from subnet 2, and 6 $(60-54)$ and 10 $(64-54)$ for the packet from subnet 3. Hence, the detection system will store in its IP-to-hopcount table hop count values of $(2, 6)$ for subnet 2 and $(6, 10)$ for subnet 3.

Based on this topology, an attacker can design the following situation. Multiple hosts can be compromised on subnet 2 with attacking agents being used that attempt to flood the detection system host with spoofed packets of initial TTL value of 64 and spoofed source IP addresses from subnet 3. All packets have an initial TTL value of 64, decremented down to 58 at the point of the detection host. Even though the attacker physically is on subnet 2, the detection system would think that the packets came from subnet 3. The physical number of hops required to go from subnet 2 to subnet 1 is 6, the same as one of the hop count values the system has stored for subnet 3 in its IP-to-hopcount table. Thus, the spoofed packet would be marked as legitimate, resulting in a false negative.

In our first experiment, when only the passive hop-count filtering was used, the detection system scanned a total of 518 packets, as shown in Table 3.1, from which 131 were spoofed. Out of those, 21 or 16% were false negatives because the detection system observed hop count values it expected, so it considered the packets legitimate.

	Scanned	**Spoofed**	**False Pos.**	**Legitimate**	**False Neg.**
Passive only	518	131	0	387	21
with IP Id	540	127	0	413	0
with direct TTL	508	126	0	382	20

Table 3.1: Minimizing the amount of false negatives.

3.2.2 Using Active Probes to Detect False Negatives

During the next experiment, we used the TTL and IP Id active probes to minimize the number of false negatives. These tests returned variable results. The IP Id probes performed superbly in this testing environment. Out of 540 total packets scanned, it detected all 127 spoofed packets as spoofed and all 413 legitimate packets as legitimate, resulting in values of zero for both the false positives and the false negatives.

Using direct TTL probes, on the contrary, the number of false negatives resulting from the detection process was almost unchanged, as illustrated in the first two rows of Table 3.1. From 508 scanned packets (126 of them spoofed), the algorithm resulted in 20 false negatives. These results are almost identical to the ones from the test where passive hop-count filtering is used alone.

3.2.3 False Negatives with Multiple Attackers

While the experiments represented situations, in which there is a single attacking host, real-world spoofed IP traffic attacks often involve multiple attackers. In order to investigate the effect of the number of attackers on the effectiveness of the detection system, we performed another set of experiments on the network topology illustrated previously in Figure 3.1.

The results from the use of a single attacking host during the spoofed attack are illustrated in Table 3.2. Out of 120 actual spoofed packets, we observed 51 or 42% false negatives. Using active Id probes in combination with the passive hop-count filtering algorithm, the system was able to detect all false negatives.

	Scanned	**Spoofed**	**Marked Spoofed**	**Marked Legit.**	**False Neg.**
Passive only	520	120	69	451	51
with IP Id	516	131	131	385	0

Table 3.2: A single attacking host.

During the second experiment we modified the network topology from Figure 3.1 by adding another attacking host to subnet 2 as illustrated in Figure 3.2. The differences in the results are negligible. As Table 3.3 shows, 183 spoofed packets resulted in 79 false negatives (or 43%), which was decreased again to 0% after the introduction of active IP Id probes.

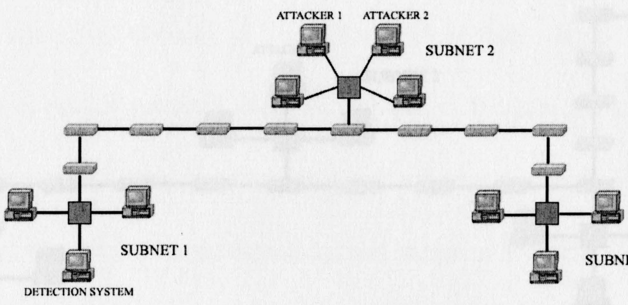

Figure 3.2: Modified network topology with two attacking hosts on the same subnet.

	Scanned	Spoofed	Marked Spoofed	Marked Legit.	False Neg.
Passive only	510	183	104	406	79
with IP Id	517	201	201	316	0

Table 3.3: Multiple attacking hosts on the same subnet.

During our last false negative experiment, we located the attackers on different subnets. For this purpose, we extended the network topology as shown in Figure 3.3 to include subnet 4, located at 4 additional hops beyond subnet 3 from the detection system. The attackers are now located on subnet 2 and subnet 3.

The results from these experiments are shown in Table 3.4. When using the passive hop-count filtering algorithm alone, 167 spoofed packets result in 73 false negatives for a false negative rate of 44%. When combining this with active IP probes, all false negatives were detected successfully.

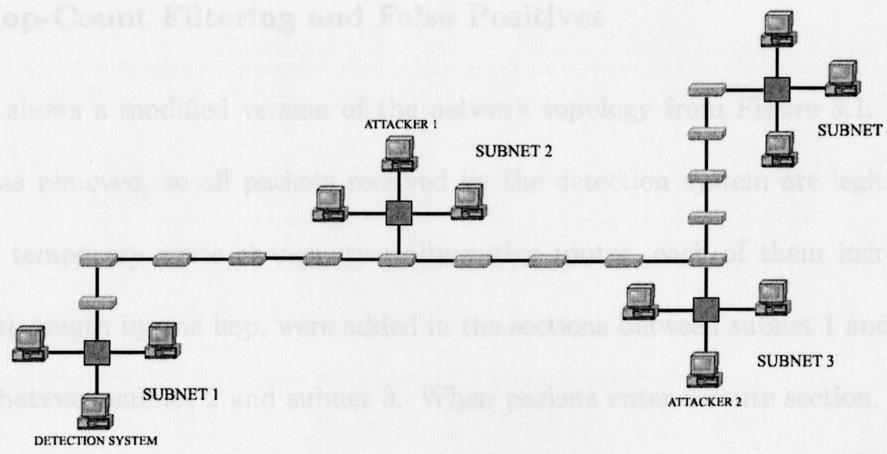

Figure 3.3: Extended network topology with two attackers on the different subnets.

	Scanned	Spoofed	Marked Spoofed	Marked Legit.	False Neg.
Passive only	511	167	94	417	73
with IP ID	544	190	190	354	0

Table 3.4: Multiple attacking hosts on different subnets.

3.3 False Positive Experiments

The passive hop-count filtering detection algorithm is susceptible to failure due to temporary route changes or Internet router failures because it records an inferred hop count at a single point of time and later uses the same value as the basis for comparison. When a route change occurs, any packets traveling along that route will result in false positives until the corresponding hop count value in the table is updated. If the duration of a route change lasts for a long period of time before the system converges to a state of stability, a large number of false positives will be flagged by the detection system.

3.3.1 Hop-Count Filtering and False Positives

Figure 3.4 shows a modified version of the network topology from Figure 3.1. First, the attacker was removed, so all packets received by the detection system are legitimate. To simulate a temporary route change, two alternative routes, each of them increasing the routing path length by one hop, were added in the sections between subnet 1 and subnet 2, as well as between subnet 2 and subnet 3. When packets enter a route section, which has an alternative route, they get forwarded along the main route 80% percent of the time and along the alternative route the remaining 20% of the time.

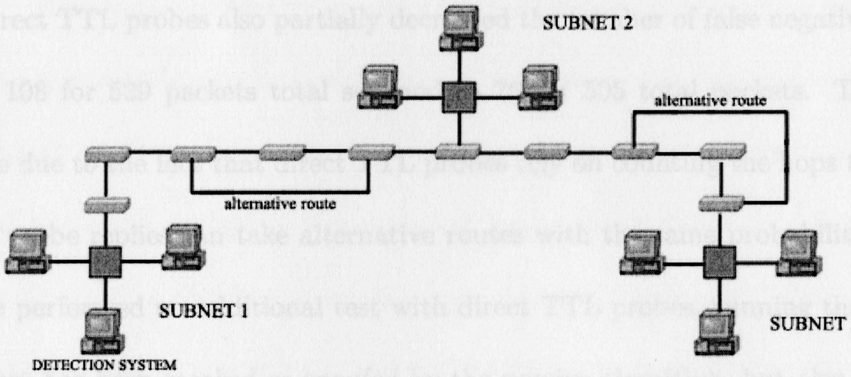

Figure 3.4: Modified network topology.

The results from these tests are summarized in Table 3.5. When using the passive hop-count filtering test only, 108 out of 529 packets scanned were considered spoofed because a change of hop counts was detected along the route. Note that the passive algorithms cannot be used to update hop count values stored by the detection system because it has no way to verify whether a hop count value is different due to a route change or due to the packet being spoofed.

	Scanned	Spoofed	False Pos.	Legit.	False Neg.
Passive only	529	108	108	421	0
with IP Id	511	0	0	511	0
with direct TTL	505	76	76	429	0
TTL (all packets)	515	147	147	368	0

Table 3.5: Minimizing the amount of false positives.

3.3.2 Using Active Probes to Detect False Positives

Both active detection methods were used to attempt to minimize the number of false positives. Active IP Id probes performed superbly detecting all route changes and reducing the number of false positives to zero.

Using direct TTL probes also partially decreased the number of false negatives, decreasing it from 108 for 529 packets total scanned to 76 for 505 total packets. These results appear to be due to the fact that direct TTL probes rely on counting the hops taken by the probes, but probe replies can take alternative routes with the same probability as normal packets. We performed an additional test with direct TTL probes, running them not only when a packet has been marked as spoofed by the passive algorithm, but also when it has been marked as legitimate. In this case, we found the number of false positives to actually increase - up to 147 out of 515 packets scanned total, as shown in the fourth row of Table 3.5. TTL probes have an advantage, however, that they can be used to update hop count values in the IP-to-hopcount table in the detection system.

Chapter 4

Conclusions

Based on our experimental results, we consider the passive hop-count filtering detection algorithm to be insufficient when used alone to provide the performance required by performance-sensitive systems in certain situations. The simulations used in the experiments represent common and real-world examples. If an attacker knows the configuration of the detection system, he can design a scenario where the weaknesses of hop-count filtering can be exploited and thus the detection system will be forced into poor performance. In addition, even though researchers have shown that routing path lengths stay relatively stable [5], changes in route lengths decrease the effectiveness of the detection system, causing it to block legitimate traffic.

IP Id probes performed quite well in both tests, but in our simulation it is assumed that routers work efficiently and provide no or little delay to packets passing through them. In real circumstances, however, real router delays along the path of a probe request can cause the probed IP identification value to be obsolete, especially for hosts with higher outgoing activity, where the identification field would be incremented by the operating system very quickly. Therefore, IP Id probes would not be as dependable, in this simulation, when applied on the real Internet.

Direct TTL probes share to a large extent the same weaknesses as the passive hop-count filtering algorithm because they base their decisions on the same hop-counting criteria. However, such probes have a small advantage over passive hop-count filtering because the value they provide is current. Therefore, if we use a topology with less frequent route changes, we expect direct TTL probes to perform better in terms of number of false positives..

Increasing the number of attackers did not have any apparent effect on the effectiveness of the detection algorithms. Thus, we can assume that these algorithms can also be used in cases when more than one attacking host is present on the network.

Our current and future work involves implementing the TCP and the ICMP protocols into the NetSim package. This will provide the detection system with more detection methods, upon which to make decisions – the three-way TCP handshake, the TCP sliding window and packet retransmission mechanisms, as well as ICMP Time Exceeded messages. All of these features have been proposed by Templeton and Levitt [21] to be used as detection methods, but there is no research available about their implementation and performance as such. The methods using features of the TCP protocol might prove to be very accurate in a variety of circumstances, but also more expensive in terms of the resources to establish a TCP connection to probe a remote host. ICMP Time Exceeded messages also have the potential for high effectiveness, but this potential might be diminished by the similar high cost in terms of resources, as well as by the fact that ICMP messages are blocked by some firewalls on the Internet.

Appendix A

The NetSim Virtual Network Simulation Package

All detection methods evaluated were implemented inside a modified version of the NetSim virtual network simulation package [15]. The package implements a number of virtual network devices – host, switch and relay - and allows them to communicate over a virtual local area network through the IP, ARP and Ethernet protocols. The spoofed IP packet detection system was implemented as an extension of the virtual host module.

A.1 Overview of NetSim

The Network Simulation API (NetSim) is a software package, which provides an object-oriented virtual simulation of a simple local area network (LAN). It provides the means to create and interconnect simple virtual host computers and network switches. It also allows for the implementation of the various Internet Protocols such as Ethernet, ARP and IP. All components of the package are represented as C++ classes, as illustrated below, which allows for easily adding new protocols and virtual hardware or modifying the existing components.

Figure A.1 shows the structure of the virtual simulation package. All components and protocols are divided according to the TCP/IP stack layer they belong to – data link, network, transport, etc. The physical layer, which provides communication between protocol stacks, is implemented through socket connections between the real hosts, on which the simulation is used. Thus, virtual Data Link layer Ethernet adaptors communicate through the socket interface of the real network hosts. The advantage of this simulation method is that it allows us to start multiple virtual network components on the same real network host, as long as each virtual component uses a different real port number. Thus, the size of the virtual network topology is not restricted by the size of the real local network used to run simulations on.

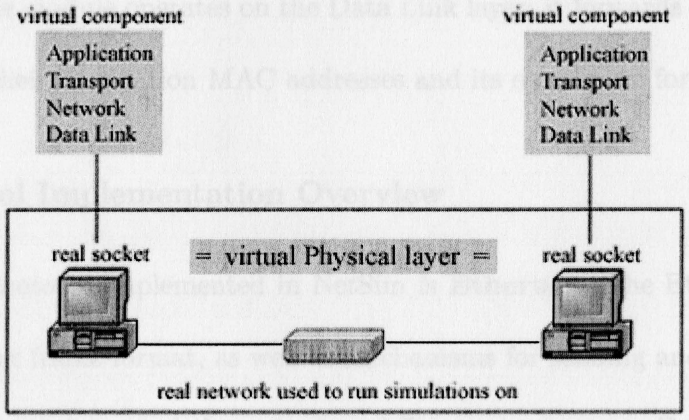

Figure A.1: Virtual simulation package on top of the real network hosts

A.2 NetSim Implementation Details

A.2.1 Module Implementation Overview

The **host module** implements a virtual host including the protocol stack. Currently, the implementation includes the Data Link, Network and Transport levels of the Internet Protocol suite. The handling of each incoming and outgoing message is handled by spawning off a child process from the main host application. The child process passes the message to the first protocol stack layer.

At the Data Link layer, the host uses virtual Ethernet adaptors implementing the Ethernet frame format and a subset of the protocol communication mechanism. The virtual Ethernet adaptors establish client connections through the socket interfaces of the real hosts used to run the simulation on.

The **switch module** waits for incoming connections from virtual hosts on a specified listening port. The module operates on the Data Link layer; it forwards incoming Ethernet frames based on their destination MAC addresses and its own frame forwarding table.

A.2.2 Protocol Implementation Overview

The lowest-level protocol implemented in NetSim is **Ethernet**. The Ethernet module includes the Ethernet frame format, as well as mechanisms for sending and receiving frames. The latter two are implemented through virtual Ethernet adaptors. Each virtual adaptor data structure contains a physical socket connection, through which the adaptor sends and receives messages.

The **ARP protocol** acts on the Network level and serves to convert IP addresses used by the high-layer protocols to MAC addresses used by adaptors at the Data Link layer. Each device's ARP service maintains a table, where it stores IP-MAC address associations. Routers are configured to drop ARP packets. If a host is unable to discover on the local network the MAC address for a destination host, it sends the message to the default gateway - the default router, which handles traffic leaving the local network.

The second protocol implemented on the Network layer is the **IP protocol**. Inside the host module, the IP protocol possibly fragments outgoing data, invokes ARP to discover the MAC address of the destination host and sends the resulting IP packet(s) down the protocol stack. For incoming data, IP assembles back all fragments and passes the message up the protocol stack.

All protocols add their own header to each outgoing message and strip that header off each incoming message before passing them up or down the protocol stack. Also, all protocols perform checksum computation and verification wherever a checksum is included in the protocol specification.

A.2.3 Modification to NetSim

The original version of the NetSim package was designed as a virtual simulation for local area networks. However, since all attacks involving IP spoofing go beyond the borders of a LAN into the global Internet, NetSim had to be extended to support global network communications. For this purpose, we implemented a router module and modified the existing host module in order to support different IP address classes and unique addressing throughout the whole simulation environment.

The **router module** behaves as a client, with one virtual Ethernet adaptor for each of its interfaces. The virtual router operates on the data link and network layers, making router decisions and forwarding IP packets through its interfaces. Each incoming packet is handled by a separate process forked from the main router application.

A.2.4 Advantages of NetSim

As mentioned above, one of the advantages of using this virtual network simulation package is the ability to run a greater number of network hosts and devices than the number of real computers used to run the simulation on. Another advantage is the modularity achieved by applying the principles of object-oriented programming and encapsulation.

On one hand, every virtual network device is implemented in its own object-oriented module, which can be compiled and run by itself. This allows for the quick reassembling of virtual device modules when an experiment requires a different network topology to be used. The object-oriented nature of the simulation also makes the interaction between each device module very clear and understandable.

Moreover, encapsulation allows for the easy reworking of each individual module without affecting its interaction with the other modules, as long as it continues providing the same interface to them. Encapsulation also makes the simulation more realistic because real-world network devices can use any implementation technique and this would not change their interaction with other devices, as long the requirements for protocol interfaces are satisfied.

Bibliography

[1] D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. *ACM SIGOPS Operating Systems Review*, 35(5):131–145, 2001.

[2] Arkin. online: http://cert.uni-stuttgart.de/archive/bugtraq/2001/05/msg00047.html.

[3] S. Bellovin. Rfc 1948 - defending against sequence number attacks, 1996.

[4] S. Bellovin, M. Leech, and T. Taylor. Icmp traceback messages. Technical report, Internet Engineeting Task Force, 2003.

[5] Haining Wang Cheng Jin and Kang G. Shin. Hop-count filtering: an effective defense against spoofed ddos traffic. In *Proceeding of the 10th ACM Conference on Computer and Communication Security*, October 2003.

[6] B. Cheswick, H. Burch, and S. Branigan. Mapping and visualizing the internet. In *Proceedings of USENIX Annual Technical Conference*, 2000.

[7] K. Claffy, T. Monk, and D. McRobb. Internet tomography. *Nature*, 1999.

[8] T. Dunigan. Backtracking spoofed packets. Technical report, Oak Ridge National Laboratory, 2001.

[9] The Swiss Educational and Research Network. Default ttl values in tcp/ip. online: http://secfr.nerim.net/docs/fingerprint/en/ttl_default.html.

[10] P. Ferguson and D. Senie. Rfc 2827 - network ingress filtering: Defeating denial of service attacks which employ ip source address spoofing. Technical report, The Internet Society, 2000.

[11] J. Ioannidis and S. Bellovin. Implementing pushback: Router-based defense against ddos attacks. In *Proceedings of Networks and Distributed System Security Symposium*, February 2002.

[12] Frank Jennings. Linux raw socket programming - what lies beneath a socket? online: http://linux.sys-con.com/.

[13] A. Keromytis, V. Misra, and D. Rubenstein. Sos: Secure overlay services. In *Proceedings of the 2002 Conference on Applications, Technologies, Architectures and Protocols for Computer Communication*, pages 61–72, 2002.

[14] R. Mahajan, S. Bellovin, S. Floyd, J. Ionnidis, V. Paxson, and S. Shenker. Controlling high bandwidth aggregates in the network. *ACM SIGCOMM Computer Communication Review*, 32(3):62–73, 2002.

[15] R. Necaise. Netsim virtual network simulation package. online: http://www.cs.wlu.edu/ necaise/software/netsim.

[16] A. Perrig, D. Song, and A. Yaar. Pi: A new defense mechanism against ip spoofing and ddos attacks. Technical report, School of Computer Science, Carnegie Mellon University, 2002.

[17] L Peterson and B. Davie. *Computer Networks: A Systems Approcach, 2e.* Morgan Kauffman Publishers, 2000.

[18] J. Postel. Rfc 792 - internet control message protocol, 1981.

[19] S. Savage, D. Wetherall, A. Karlin, and T. Anderson. Practical network support for ip traceback. In *Proceedings of the Conference on Applications, Technologies, Architectures and Protocols for Computer Communication*, pages 295–306, 2000.

[20] The Internet Society. Rfc 791 - internet protocol, 1981.

[21] S. Templeton and K. Levitt. Detecting spoofed ip packets. In *Proceedings of the Third DARPA Information Survivability Conference and Exposition*, 2003.

[22] Ju Wang and A. Chien. An analysis of using overlay networks to resist distributed denial-of-service attacks. unpublished.

[23] Wikipedia. Internet protocol suite. online: http://en.wikipedia.org.