I, Adam M. Neiman, hereby submit this work as part of the requirements for the degree of:

Masters

in:

Computer Science

It is entitled:

Hash Stamp Marking Scheme for Packet Traceback

This work and its defense approved by:

Chair: Dr. John Franco

Dr. Dharma Agrawal

Dr. Qing-An Zeng
Hash Stamp Marking Scheme For Packet Traceback

October 14th, 2005
Adam Neiman

B.S., Computer Science & Engineering
M.S., Computer Science

Department Of Electrical and Computer Engineering, and Computer Science
College Of Engineering

Committee Chair: Dr. John Franco
Abstract

The Internet Protocol (IP) is the basic language that all computers use to communicate across networks and the Internet. A flaw in the design of this protocol allows attackers to forge the sending address of IP packets, known as packet spoofing. This packet spoofing is a serious security issue on networks and the Internet because it prevents authorities from locating the true source of any spoofing attack. In this paper we analyze technologies available for coping with packet spoofing. After this discussion we present a simple method for traceback, followed by an analysis of the method's requirements.

Keywords

hash, message authentication code, MAC, hash-keyed message authentication, HMAC, Internet Protocol, IP, spoofing, packet stamping, packet marking, traceback, computer network security, denial of service, DoS
Acknowledgments

I would like to thank my parents, Philip and Patricia, for their love, support and guidance during my trials of researching and writing this paper. I do not think this would have been possible without their constant listening to my thoughts and pushing me to think harder and to think “outside of the box.”

I would like to thank my sister, Ashley, for her help with designing the diagrams and pictures found in this paper. Her input was key to designing understandable layouts and graphics.

I would like to thank my friends Jason Liedigk and David Champnoise. My two best friends were always asking how my thesis was going and if I had made any progress getting the paper done, or if I had figured out a solution to a rut I had gotten stuck in.

I would like to thank my ex-boss from the University of Toledo, Gregory Gaustad. He set aside a portion of his time at work to listen to my ideas and critiqued my ideas from a business point-of-view. Mr. Gaustad encouraged me to think creatively and he gave me my first experiences with security.

Last, but not least, I would like to thank Dr. John Franco for being my adviser. My classes and discussions with him were always an inspiration and served as a catalyst to bring out some of my best efforts and achievements. Besides helping me to broaden my horizons, Dr. Franco helped me realize, through his enthusiasm about security, my own interest in and enthusiasm for computer and network security as a profession.
Table of Contents

Abstract 3
Keywords 3
Acknowledgments 5
Introduction 8
The Problem 11
Assumptions 11
Background 13
  Internet Protocol 13
  Per Packet Storage Space 14
  Detecting Spoofed Packets 16
  Spoofing Attacks 17
Previously Proposed Solutions 19
  Probabilistic Packet Marking 19
  Message Digest Logging 21
  ICMP Packet Messages 22
  Pattern Filtering 22
  Link Testing 23
  End-to-End Encryption 24
Proposed Solution 25
  Packet Traffic Information 28
  Hash-keyed Message Authentication Code 30
  Stamp 31
  Traceback 33
Resistance To Attacks 39
  False Negatives 39
  Extraneous Information 40
<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Proof Of Correctness</td>
<td>41</td>
</tr>
<tr>
<td>Analysis</td>
<td>44</td>
</tr>
<tr>
<td>Bibliography</td>
<td>53</td>
</tr>
<tr>
<td>Appendix A: Brute-force Attack Test Program Source Code</td>
<td>61</td>
</tr>
<tr>
<td>Appendix B: Definitions And Terms</td>
<td>68</td>
</tr>
<tr>
<td>Index</td>
<td>69</td>
</tr>
</tbody>
</table>
Introduction

In this paper we propose a solution to the problem of tracing spoofed Internet Protocol (IP) packets back to their true source. IP spoofing is the number one security issue on the Internet. (Meijer, 2001) This spoofing occurs when an attacker forges the sending address of IP packets in order to make it appear the packets originated from a different host.

IP spoofing is a serious security issue for several reasons, primarily because it allows an attacker to remain anonymous. First, in absence of a mechanism for calculating the true path of a packet, authorities are unable to stop spoofed attacks, which could last several hours, if not days. Second, attackers are able to use spoofing to convince a victim it is talking to another host. In the case of a trusted host, an attacker could gain privileged access to a remote computer without first having to follow all required steps for authentication. A trusted host is defined as: “A computer not required to go through full authentication to gain access to your computer.” (“Glossary”, 01/2002) Last, spoofing can be used to implicate an innocent host in a spoofing attack. Attackers could initiate an attack against a host using packets spoofed with an innocent host's address, causing authorities to penalize the innocent party.

Spoofing attacks fall into two main categories: spoofing with packet sniffing and spoofing without packet sniffing. (Meijer, 2001) Packet sniffing is defined as: “the act of capturing packets of data flowing across a computer network....Packet sniffing is to computer networks what wire tapping is to a telephone network.” (Bradley, 2005) Spoofing is commonly used with packet sniffing to trick a victim into believing it is communicating with a desired party, commonly known as connection hijacking. Common examples of this communication hijacking include DNS spoofing, ARP spoofing, and TCP connection hijacking. Spoofing attacks that do not use sniffing are commonly used as one-way attacks to exhaust resources or to mask malicious communications. Common examples of this include denial-of-service attacks and stealth scans.
Previous attempts to solve or prevent IP spoofing include probabilistic packet marking (Savage, 2001; Dean, 2002; Goodrich 2002), logging message digests of packets (Song, 2001; Snoeren, 2002), pattern filtering (Jin, 2003; Tupakula, 2003; Bremler-Barr, 2004), ICMP packet messages (Bellovin, 2003), link testing (Burch, 2001), and end-to-end encryption. The first two approaches, probabilistic packet marking and digests logging, attempt to provide a framework for locating the true source of spoofed packets. The remaining approaches do not provide any such mechanism for traceback, but attempt to prevent substantial amounts of spoofed packets from reaching the victim. All of these attempts have major drawbacks which have prevented partial or total implementation on the Internet.

These attempts at a solution for IP spoofing failed on some aspect, whether it be a feasibility or incentive issue. Under the following restrictions, these solutions fail to prevent IP spoofing and/or provide traceback functionality.

1. Minimize changes to router hardware, software, or firmware.
2. Minimize additional burden on routers caused by the traceback process.
3. Minimize the added congestion (if any) on any individual links involved in the traceback process.
4. Provide the ability to trace individual packets.
5. Minimize the computational burden on the victim.

The probabilistic packet marking scheme fails the fourth and fifth restrictions. The message digest logging technique fails to meet the first restriction. The ICMP messaging method fails to satisfy the third and fourth restrictions. The pattern filtering scheme fails to satisfy the second and fourth requirements. The link testing scheme fails to meet the third restriction. The end-to-end encryption scheme fails to satisfy the fourth requirement.
The proposed solution solves the problem under all five restrictions. First, our solution relies primarily on cryptographic hashing and cryptographic hash-keyed message authentication codes. According to the data sheet for the Cisco VPN Acceleration Module for the 7000 Series Cisco VPN Routers ("VPN Acceleration Module for Cisco 7000 Series VPN Routers", 2004), it is possible to provide MD5 and SHA-1 digest capability using router modules. Implementing hashing and hash-keyed message authentication code (HMAC) functionality on hardware also minimizes the burden placed on each router's processor when compared to a software-only solution. Our solution avoids adding congestion by restricting traceback requests. Each traceback request is forwarded if and only if the current router handled the original packet within an acceptable certainty. The amount of work done by the victim during an attack is minimized because it no longer has to analyze abstract data to form a coherent attack path (i.e. probabilistic packet marking).

An analysis of stamping proposals reveals two distinct ideas: storage of information on the router (Snoeren, 2002) and storage of information in packets (Savage, 2001; Dean, 2002; Song, 2001; Doeppner, 2000). Each proposal provided valuable insight into tracking spoofed IP packets, but no proposal seemed to address storage both on the router and in packets. To my knowledge no research into this idea exists, and I have no evidence as to why this idea was not considered and/or researched. The best information regarding routers from that time period is that they had less on-board memory. Today's routers commonly contain 128 to 512 megabytes of memory, with the higher-end routers, commonly used on the Internet, containing 512 megabytes. 10 megabytes is only a fraction of this memory size, about 2%, compared to 8% of 128 megabytes and 16% of 64 megabytes. Also, the router industry is very protective of router information and in most cases will not release any information. Without any valid specifications on the amount of free memory available, researchers were and are more likely to research other methods.
The Problem

An intruder may send packets that spoof other IP addresses in order to prevent a victim from uncovering the actual location from which the packets were sent. With current technology, it is difficult to find that location. This is partly because there is very little space in a packet that is available for traceback and partly because the amount of information that must be stored in that space is too great when using a naive method. The problem is: with minimal changes to current technology and providing a high incentive for use, propose a scheme that allows, with high certainty, traceback of individual IP packets.

By minimal we mean:

1. The average size of a packet should not increase to where it significantly inhibits the transmission speed across links (between routers).
2. The processing burden of any router should not increase to where it significantly inhibits the router's capability to process and route packets.
3. The traceback process should not add to the congestion on any link by significantly increasing the number of packets transmitted across that link.
4. The traceback process should not require any extensive and/or expensive modifications to existing hardware.

Assumptions

The following assumptions are based on current router technology and configuration:

1. **Attackers are aware of the traceback system and may attempt to interfere.**

   According to the article “Cyberterrorism Hype,” sophisticated attacks against hardened targets requires extensive knowledge in programming, systems administration, networking theory, and networking security, along with a large amount of free time. Attackers will do anything within their abilities to prevent authorities from tracing their spoofed packets. The attackers we are most concerned
with will have extensive knowledge in programming, networking, and security. These attackers also will research any data published about our traceback methods. Note that the attacker is also able to interfere by initiating traceback requests and/or by faking traceback request packets.

2. **An attacker is unlikely to attack and subvert a router.** “Hackers have hitherto concentrated most of their fire on the servers, hosts and private networks attached to the Internet, rather than on the underlying IP infrastructure....end systems constitute a more rewarding target.” (Hunter, 2004, p. 11)

3. **Links only connect routers, except possibly for the link back to the attacker or victim.** “On the Internet, routers are usually connected to the next hop routers through physical links.” (Ohta, 1997)

4. **All packets go through a router one time.** “…the same functions of traditional routers, routing each packet individually…” (Twiggs, 2001)

5. **There is sufficient space in IP packets to store router stamps.** There are specific IP header fields (i.e. TOS and ID fields) which we can overload to store router stamps. (Dean, 2002, p. 134; Song, 2001, p. 2)

6. **There is a sufficient amount of physical memory available for a minimal lookup table.** A minimal amount of memory is essential to the operation of the traceback scheme introduced in this paper. Currently I do not have access to specific data about the memory utilization of existing backbone routers.

7. **Any hash-keyed message authentication code (HMAC) or hashing capability either already exists on the router or can be incorporated by the use of a router module card.** According to the data sheet for the Cisco VPN Acceleration Module for the 7000 Series VPN routers, modules are capable of supporting cryptographic hashes, specifically the SHA-1 and MD5 digests. (“VPN Acceleration Module for Cisco 7000 Series VPN Routers”, 2005)
8. **Some responsible entity initiates the traceback sequence.** This entity could be a network/system administrator or program, but does not include typical users. 

“Computer security is too complicated and the bad guys are too devious and inventive....It’s simply unrealistic to assume that average users can keep up with them.” (Prokop, 2005)

9. **A public key infrastructure (PKI) is available for routers signing traceback requests.** This allows the victim to authenticate and confirm the integrity the results of a traceback request.

**Background**

**Internet Protocol**

The Internet Protocol is the *de facto* standard used for communication across the Internet. “...TCP became the de facto standard over which the Internet and most of the actual intranets and extranets rest.” (de Vivo, 1999, p. 83) It is used as the underlying protocol for applications including instant messaging, web browsing and email. (Oliver, 2005)

Communication using IP involves sending a stream of IP packets to a specific host. “Data transmitted over the internet using IP is carried in messages called IP datagrams.” (Kozierok, 2004) Each packet is divided into two basic sections: the headers and the data. The headers contain all the information needed to deliver the packet including the version of IP being used, the destination's address, the sender's address and a number of other items. The data portion contains the message that is being delivered. See “IP Datagram General Format”, (Kozierok, 2004) table 56 for a detailed description and size of each header field.

As IP packets leave the *true sending host* and local network, they encounter routers. These network devices connect two or more networks and allow packets from a particular network to travel on to and through an adjacent network. When a router processes an individual packet, it compares the packet's destination with the contents of a local routing table. The router then sends the packet off in the direction it determines is most beneficial.
This style of routing may appear sufficient, but its weaknesses become apparent when IP spoofing is encountered. IP spoofing occurs when individuals forge the address of the sending host in IP packets. The current style of routing does nothing to ensure the validity of the sending address marked in each packet. “The packet's 'Source IP' is only used when it finally arrives at its destination.” (Gibson, 2003)

IP spoofing creates a number of problems for networking and system administrators. Malicious viruses and worms, along with hackers, attack computers daily. IP spoofing allows for anonymous attacks because the packets involved cannot be traced back to their true source. IP spoofing is most commonly used in a number of network attacks including denial of service (DoS) attacks, distributed denial of service (DDoS) attacks, and the smurf attack. It also allows individuals to easily defeat host-based authentication and cripples the use of remote access tools like telnet and rsh.

**Per Packet Storage Space**

Out of the 160 bits that make up the header fields of each IP packet, few bits are usable for storing a router stamp. Most of the fields are actively used and cannot be modified without a negative impact on IP routing. Currently information is not available on how specific routers handle packet routing. As a result I do not have statistics which could be used to determine which fields are not statistically used in normal routing. However, the “Version”, “Internet Header Length”, “Total Length”, “Time To Live”, “Protocol”, “Header Checksum”, “Source Address”, “Type of Service”, and “Destination Address” fields do not appear to be modifiable without significant effects on routing. This leaves us with approximately 50 out of 160 bits for the router stamps.

According to an OC48 Trace (Ma, 2002) provided by the Cooperative Association for Internet Data Analysis (CAIDA), fragmented IP packets were found to comprise 2,698,606 of the 334,177,587 IP packets measured (direction zero), or approximately 0.8%. With precedence (Song, 2001; Dean, 2002; Savage, 2001; Goodrich, 2002), we can consider
this a representative of the proportion of fragmented packets on the Internet. As a result, we can ignore fragmentation of IP packets and make use of fields related to fragmentation. The remaining fields we can work with are:

1. **Identification**: This serves as a unique identifier for a group of fragments belonging to the same original unfragmented packet. This should be usable due to the fact that we no longer need to associate a single fragment with a group of packet fragments.

2. **Fragmentation Offset**: “When fragmentation of a message occurs, this field specifies the offset, or position, in the overall message where the data in this fragment goes.” (Kozierok, 2004) This field is no longer needed.

3. **Flags**: This field is comprised of 3 bits: *Reserved, Don't Fragment, and More Fragments*. The *Reserved* bit is reserved for future use and could conceivably be used for storing a portion of the router stamps. The *Don't Fragment* bit is used to control whether or not fragmentation takes place when needed. Since we are ignoring fragmentation, the value of the field should not have any effect over routing. The *More Fragments* bit indicates whether or not the current fragment is the last fragment to be sent. I do not have information describing how specific router brands/models handle this bit.

Using the **Identification** field, **Fragmentation Offset** field, **Reserved** bit, and **Don't Fragment** bit, we have 31 bits to use for storing router stamps. The MD5 message digest algorithm typically outputs digests of length 128 bits or longer. (“What are MD2, MD4, and MD5?”, 2005) Compared to this length, 31 bits is clearly insufficient for storing a full-length cryptographically secure message digest.
Detecting Spoofed Packets

Mechanisms used to find spoofed packets attempt to detect abnormal patterns in packets received by a host. Techniques for detecting spoofed packets fall into two main categories: those used on routers and those used by end systems. Router-based techniques examine packets for anomalous activity when packets leave or enter a local area network (LAN). This might include using techniques like ingress filtering. This type of detection might not be useful because the victim relies on the router(s) to notify it of an attack. Host-based techniques typically involve comparing packets against established normal values, treating anomalous packets as spoofed. Host-based methods are divided into two main categories: active and passive.

Active methods rely on the fact that spoofing attacks are blind attacks. “Active methods either make queries to determine the true source of the packet (reactive), or affect protocol specific commands for the sender to act upon (proactive)” (Templeton, 2003, p. 4) When using a blind attack, the attacker must make an educated guess for his or her replies because the attacker does not receive feedback from his/her spoofed packets. Active methods attempt to foil this by requiring interactivity with the sending host(s). If the sending host cannot be contacted (i.e. host non-existent or is heavily firewalled), the packet is likely to be spoofed. In the case the sending host is alive and communicating, the goal is to solicit a response and compare the values from the response to that of the packet(s) in question. If the packets in question appear similar to the interrogation response, the packet is most likely not spoofed. Another way to test for interactivity is to start a communication handshake with a zero-length window size, for example with the Transport Control Protocol (TCP). (Templeton, 2003, p. 5) If the sender continues to send packets, it is obvious that the sender is unaware that the window size is currently zero and packets received from it are likely to be spoofed.

Passive methods include analyzing packets received from a particular host and establishing a set of normal values. “Where observed data will have a predictable value...we
can learn what values are expected and consider packets with unexpected values suspicious.” (Templeton, 2003, p. 6) Common examples of consistent packet properties include the time-to-live (TTL) and the type of service (ToS) header fields. Packets coming from a particular host tend to have roughly the same TTL value, given the fact that they tend to travel along the same path over a relatively short length of time. In a number of cases, passive techniques are supplemented with active methods to test for spoofing. Packets marked as spoofed by the passive method would then be tested by using an active method (i.e. testing to see if the sender is alive). “Rather than use passive methods alone, by using them in combination with reactive methods we can construct an efficient spoofed packet detection system.” (Templeton, 2003, p. 6)

**Spoofing Attacks**

“A key factor in all packet-spoofing attacks is that it is not necessary for the attacker to directly receive packet replies from the target. Replies are either unimportant, their contents can be inferred, or the packets can be observed in transit.” (Templeton, 2003, p. 2) Spoofing attacks include, but are not limited to: man-in-the-middle attacks, stealth scans, denial-of-service attacks, and reflector attacks.

Man-in-the-middle attacks typically occur when a malicious user transparently intercepts communication between two hosts. “In these attacks, a malicious party intercepts a legitimate communication between two friendly parties.” (Tanase, 2005) The attacker then uses a combination of spoofing and packet sniffing to trick both hosts into thinking the attacking host is the other side. The attacker is then able to control communications and modify or delete information sent, without the knowledge of either the sender or recipient. “In this way, an attacker can fool a victim into disclosing confidential information by 'spoofing' the identity of the original sender, who is presumably trusted by the recipient.” (Tanase, 2005)
Port scanning is often used by attackers to profile a victim before an attack. This method involves systematically probing a series of ports on a victim to determine whether or not they are open or closed. Attackers use open ports to detect services (i.e. web server, ssh server, database server) which are running on the victim, as well as the victim's operating system. In a number of circumstances, it is beneficial for the attacker to hide his/her packets in order to avoid alerting administrators of a pending attack. In a number of cases a solitary port scan coming from a single host could indicate a pending attack. To avoid this the attacker could send a stream of spoofed scanning datagrams, effectively hiding his packets from plain view. “By hiding the actual scan in a large amount of spoofed scanning datagrams from a wide range of IP addresses, the attacker will be able to hide the real scan from the administrators.” (Meijer, 2001)

Denial-of-service (DoS) attacks are the most difficult spoofing attack to defend against. “IP spoofing is almost always used in what is currently one of the most difficult attacks to defend against – denial of service attacks, or DoS.” (Tanase, 2003) “In a Denial of Service (DoS) attack, the attacker sends a stream of requests to a service on the server machine in the hope of exhausting all resources like 'memory' or consuming all processor capacity.” (Najmi, 2002) The most common DoS attack is known as the SYN flood attack. The attacker takes advantage of TCP handshaking and sends a large number of TCP SYN packets to a victim. The victim sends a TCP ACK packet back to the spoofed address and waits for a reply. Since the attacker in DoS attacks spoofs non-existent or unused addresses, the spoofed address never replies and the victim times out while waiting for a response. This waiting ties up a buffer on the victim. When all these buffers are in use by the attack, network communication is no longer possible and legitimate clients are denied access. After a preset length of time the victim releases the buffer. However due to the massive amount of attack packets released by the attacker, the buffer is more likely to be allocated for an attack packet than it is a legitimate client.
The third type of attack is known as a reflector attack. In this type of attack the attacker sends specially-crafted packets to a network host before they are sent on to the victim. One such attack is known as the smurf attack. In a smurf attack, the attacker sends spoofed ICMP echo packets to the locale subnet broadcast address, causing alive hosts to send a response to the source. This results in a significant degradation in network performance. Also, reflector attacks are commonly used in DoS attacks. In DoS attacks, the attacker(s) send packets to intermediate hosts which are spoofed with the address of the intended victim. These intermediate hosts then send their response to the victim. From the victim's point of view, the attack packets were originally sent by the innocent intermediate hosts. (Kumar, 2005)

Previously Proposed Solutions

A number of technologies have been developed in recent years to combat IP spoofing. While some differ in method, they all have the same goal: develop a method capable of tracing spoofed packets back to their true source and/or prevent spoofed packets from reaching the victim. These methods include probabilistic packet marking, message digest logging, ICMP packet messages, pattern filtering, asynchronous system (AS) keys, link testing, and end-to-end encryption. In most cases, these technologies have not been uniformly implemented on networks due to feasibility issues. In some cases they require changes to infrastructure that are too extensive, require purchasing expensive and dedicated hardware, or simply did not provide enough incentive for installation.

Probabilistic Packet Marking

One of the first notable attempts at traceback involves probabilistic packet marking and was developed by Stefan Savage. In this scheme, routers encode edges of the attack path into each packet. Each mark consists of the start field, end field, and a small field used to represent the distance of the edge from the victim. Each router that handles a given packet
probabilistically chooses whether or not to mark the packet. If the router decides to mark the packet, the router encodes its address in the start field and encodes a zero into the distance field. If the distance field was already zero, this indicates a previous router has already marked the starting point of the edge and the router encodes its address as the ending address and increments the distance field. If the router does not encode its address into the start or end fields, it still increments the distance field. Using a number of combinatorial operations, the victim is able to piece together the attack graph, given the edge information found in packets it receives during an attack. This method of spreading out information across a group of packets does not allow the victim to traceback an individual packet, which is a violation of our requirements. For more details on the algorithm used during this phase, see Figure 4 on page 231 from “Network Support For IP Traceback”, Savage. et. al.

The first limitation of this method is that it requires a high computational overhead for the victim, which is a violation of our requirement to minimize the amount of work done by the victim. Second, this method also produces a large number of false positives when solving a distributed attack. “…this approach can require days of computation to reconstruct the attack paths and gives thousands of false positives even when there are only 25 distributed attackers.” (Song, 2001, p. 1) Also, this method does not authenticate any stamps, and is vulnerable to forging by compromised routers. “If a router is compromised, it can forge markings from other uncompromised routers...the victim will not be able to tell a router is compromised just from the information in the packets it receives.” (Song, 2001, p. 1) Last, this method assumes there is no bad information (i.e. noise) involved in any of the packet markings. For details of a simulation using this scheme, see section c, “Simulation Results” in “Advanced And Authenticated Marking Schemes For IP Traceback”, Song et. al.

In May of 2002, a new method was designed which took advantage of Savage's work and used algebraic equations, followed by a combinatorial step, to reduce the significant overhead of the victim. This approach models path information as points on a polynomial. In simple cases where all packets arrived from a single host and via a single path, the
calculations would be a set of linear equations. In other cases, “more sophisticated interpolation strategies that succeed even in the presence of incorrect data or data from multiple paths...” (Dean, 2002) These sophisticated methods tend to be complex and work poorly with distributed attacks. “Moreover, the interpolation-with-noise algorithms are complex and slow for large distributed denial-of-service attacks...” (Goodrich, 2002) This is a violation of the requirement to minimize the amount of work done by the victim during traceback.

**Message Digest Logging**

Another scheme involves logging message digests of packets at each routers. In this system each router uses a hash to create a message digest of each packet as it is received. These message digests are stored locally for a predetermined length of time. This system is very secure in that the message digests do not reveal any part of the original packet and therefore maintain confidentiality. This idea does require routers to have a significantly large external hard drive present. In cases of Internet backbone routers, the storage space needed for a minimal length of time (i.e. 5 minutes) could easily require as much as 1 terabyte.

Currently terabyte hard drives are still rather expensive and tend to have slow average seek times. Hitachi offers a 500GB hard drive (7k500 Series) for around $520. A 1 TB hard drive could cost at least twice as much as the 500 gigabyte hard drive. For an ISP maintaining 100 routers, this would easily cost at least $100,000. This price does not include the costs incurred by installing or maintaining the hardware or any costs of router modifications. This type of expenditure is unlikely, given the decreased budgets of typical IT departments. (“2005 IT Budget Picture Still Fuzzy”, 2005) Clearly adding an expensive dedicated hard drive to each router is a violation of the requirement to minimize changes to routers.

Another issue with larger hard drives is latency and seek times. The 7k500 series drives are 7500 revolutions per minute (“Deskstar 7k500 Datasheet”, 2005), which translates to a 8.5 ms seek time. Smaller drives feature 15,000 RPM and approximately a 4.3
ms seek time. When compared to the 15,000RPM drives, the 7500RPM drive could add a significant delay to the traceback process.

**ICMP Packet Messages**

The next scheme involves routers sending additional ICMP packets when it receives arbitrary IP packets. Much like the probabilistic packet marking scheme, the goal is to “mark” packets probabilistically. In this case, the mark is passed along in a modified ICMP packet. Based on a predetermined probability, an arbitrary router sends the destination an ICMP packet which identifies the router and part of the IP packet's contents. In cases of flooding attacks, the victim is able to puzzle together the attack path by piecing together the ICMP messages received.

This scheme has a number of drawbacks. First ICMP is a connectionless protocol, and there is no guarantee that these router messages will reach the victim. If the network is heavily congested some messages could get dropped and thus prevent the victim from calculating the true source of the attack. Second, this scheme depends on input debugging capability, which may or may not be available on routers. Third, in a number of cases ICMP is often blocked and/or filtered by networks when they are under attack. “ICMP traffic is increasingly differentiated and by itself be filtered in a network under attack...” (Savage, 2001) Note that this technique violates three requirements: minimize congestion added to any link, allow for the traceback of individual packets, and minimize the amount of work done by the victim during traceback.

**Pattern Filtering**

Pattern filtering is one of the most commonly used methods of mitigating spoofed packets. Two examples of pattern filtering include ingress and egress filtering. In both cases the edge routers check the sending address of IP packets when they enter or leave an asynchronous system (AS). With ingress filtering, edge routers check packets entering the
local AS. If the sending address pretends to belong to the local AS but really comes from a non-local AS, the packet is dropped. In egress filtering, the edge routers check packets leaving the local AS. If the sending address does not belong to the local AS, the packets are dropped.

Pattern filtering on routers has the potential to be very effective in combating spoofing attacks that span multiple networks. Ingress filtering is commonly used to guard against being the target of a spoofing attack based in another AS. This process fails to stop incoming packets which are spoofed with a remote address. Egress filtering is used to prevent being the source of a spoofing attack against an address in another AS. This however does not stop outgoing packets which spoof a local address. Also, the processing burden of pattern-filtering routers is dramatically increased because the routers must examine all packets in detail. This is a clear violation of the requirement to minimize the amount of work done by routers during traceback and/or attacks. In some cases using pattern filtering could require the purchase of new hardware. “The main disadvantages of this approach include the high overhead and the associated decrease in performance....It may require upgrade of existing hardware as not all routers support this type of filtering.” (Tupakula, 2003, p. 3)

**Link Testing**

During an attack, network administrators attempt to recreate the attack path from a victim by using the chargen service to incrementally flood links between routers and the victim. If the flooding causes the attack on the victim to diminish, the link is recorded and flooding occurs between that router and its peers. This is repeated until the attack path is complete.

This technique is useful in limited circumstances, but has several drawbacks that inhibit its use. First, in cases where an attack is severely congesting a particular link, flooding that link might not produce any discernible results and could cause an administrator to believe the link is not a part of the attack path. Second, this method is only usable during an active
attack and no tracing can be done post mortem. Third, in cases of distributed attacks, this method could require lengthy amounts of time in order to locate all the attackers. Also, because the victim must coordinate its testing with upstream hosts, it must have a fairly extensive map of the local and surrounding networks before attempting to trace an attack. Last, a number of ISPs are disabling the chargen service. This requires victims to locate upstream hosts which are willing to produce the require load needed to test each link. “We recognize that ISPs are now quite regularly turning off the services we exploit to induce these loads. Thus, we must identify cooperative hosts at the right places...” (Burch, 2001, p. 1) Note that this method could potentially add a significant amount of congestion to links, a violation of requirement 3, and requires a lot of manual testing of individual links, a violation of requirement 5.

End-to-End Encryption

End-to-end encryption is commonly used in a number of applications like SSL. This method is highly secure and includes checks to prevent breaches in confidentiality and message integrity. While using encryption is a good precaution, this approach is typically very slow in comparison to cryptographic hashing and plain-text. The first issue with this approach is that both ends must have the same ability to encrypt and decrypt the messages. This type of encryption does not protect all outgoing or incoming communication. One solution to this problem is to use a virtual private network (VPN). Built with Internet Protocol Security, or IPSec, this technology forms a private, encrypted network between two network entities. This technology is not entirely compatible with IP version 4 or Network Address Translations (NATs) commonly found in a number of routers. “Currently IPSec does not support Network Address Translation (NAT), and therefore the IPSec connection cannot traverse NATed environments.” (Taylor, 2002)
**Proposed Solution**

Traceback can be accomplished by means of a stamp, which contains information about the links used to traverse the path from the *true sending host* to the victim, in conjunction with information archived on each router. This stamp, when decoded, will allow a victim's traceback packet to traverse the *true path* back to the *true source* of a spoofed IP packet. (See Figure 1.) This stamp consists of a small number of bits, occupies currently unused packet header space, and is immutable by all parties except for routers. Stamps are considered immutable because they are created through a process which involves a secret key, a hashing function, and a hash-keyed message authentication code. The stamp is required to be a part of every IP packet, except for traceback packets, sent through the network. Each router on the path an IP packet takes through the network contributes traceback information about when, on what link, and from whom the packet was received. The combination of these contributed values from each router correspond to one and only one unique path. These encoded values are unique to a specific router such that an arbitrary combination of the values can be matched to only a single router. Thus during traceback, only a single router will be able to identify itself at each step as having handled the original errant packet. Any other router receiving the traceback packet will ignore the packet.
Figure 1. Shows how the destination, source, and spoofed source (spoofed host) of a spoofed IP packet are connected. The true source of the packet is connected to the destination by \( n \) different routers. The spoofed host is connected to the destination by \( m \) different routers, some of which could be in the path from the true source to destination.

Traceback commences from the victim with transmission of a traceback packet containing the stamp taken from the attack packet. This action should be performed within two minutes of receiving the original spoofed packet, in order to provide sufficient time for
traceback. The traceback packet will then jump from router to router, in reverse order, along
the path taken by the attack packet. At any single step in the traceback process, a router
could forward a copy of the traceback packet to more than one router, but only a single copy
will continue. All routers store information when a packet passes thought it, and upon
receiving a traceback packet will examine a portion of that information and information
obtained from the stamp embedded in the traceback packet. Each router on the path will use
the stamp information provided by the traceback packet to determine the next link in the
traceback path. From this examination, only one router will be able to identify itself as the
handler of the original attack packet. That router will continue the traceback process: that is,
compute a link, digitally sign the appropriate field of the traceback packet, and send copies
of the traceback packet along that link to other possible routers. All other routers will drop
the traceback packet. Eventually the link back to the attack packet's point of origin is
identified and information regarding the true source is returned to the victim.

In order to make this method practical, we will make use of a small amount of
information stored at each router, a small-sized stamp stored in each packet, and require that
only one copy be allowed to continue the traceback at any point. The proposed traceback
scales well because each time a router identifies itself as a handler of the original attack
packet, only a single copy of the traceback packet is allowed to continue. Each time a router
determines it processes the traceback packet and forwards one or more copies, but a single
router responds. Because packets are grouped together, many will index into the same table
location. This indexing allows us to compress the table size and significantly lower the
amount of memory we will need. The trick will be to find an efficient indexing scheme that
allows for minimal memory and bandwidth usage at each router.

The obvious implementation, whereby the stamp is an index into a router table and
specifies a link address, is not efficient since one table entry would have to exist for each
packet going through the router. To decrease the size of the table, we must somehow group
the packets to reduce the number of entries needed. Grouping by the receiving link and the
previous-hop router does not reduce the size of the table enough to be economical. However, an implementation based on previous-hop router, receiving link and approximate arrival time, or time interval, would be feasible. Each router lookup table indexes on part of a checksum on packets grouped by the above method. Each entry in these lookup tables contains a key\textsubscript{previous} to a lock\textsubscript{previous} from the previous-hop router. The purpose of this recorded lock\textsubscript{current} and key\textsubscript{previous} is to affirm that a particular attack packet, carrying key\textsubscript{previous}, was received on a particular link, from a particular router during a particular time interval, identified by the lock\textsubscript{current} key\textsubscript{current} opens.

The remainder of this section details the lock, Packet Traffic Information (PTI) and Packet Identification Number (PIN) stored at each router, and how to use this information to compute a traceback link.

**Packet Traffic Information**

The Packet Traffic Information (PTI) for a particular packet is a message digest that is intended for use by routers to uniquely place an attack packet at a particular router, entering on a particular link, during a particular time interval. A link is defined as a physical connection between two network entities (i.e. routers, computers, switches). Each PTI has two components called link and router. The link component identifies the receiving link, and the router component identifies the router from which the packet was received, known as the previous-hop router. Each PTI digest is formed using the time interval of arrival as a salt. “In cryptography, salt consists of random bits used as one of the inputs to a key derivation function. A salt is typically used in a hash function.” (“Salt”, 2005) A PTI is the output of a hash function, such as MD5 or SHA-1, after it has been updated with the time interval, the media access control (MAC) address of the receiving link, and the MAC address of the previous-hop router. The following is Java code for calculating the PTI of a packet:

```java
public String generatePTI(String previousMAC, String receivingMAC, String receivingInterval) {
```
try {
    MessageDigest md = MessageDigest.getInstance("MD5");

    /***
    * Encode the arrival time interval into bytes using utf-8 and
    * update with it
    ***/
    byte[] utf8 = receivingInterval.getBytes("UTF8");
    md.update(utf8);

    /***
    * Encode the receiving link's MAC into bytes using
    * utf-8 and update with it
    ***/
    byte[] utf8 = receivingMAC.getBytes("UTF8");
    md.update(utf8);

    /***
    * Encode the previous-hop router's MAC into bytes
    * using utf-8 and update with it
    ***/
    byte[] utf8 = previousMAC.getBytes("UTF8");
    md.update(utf8);

    /***
    * Finish the MAC computations and output the digest
    ***/
    byte[] digest = mac.digest();

    /***
    * If desired, convert the digest into a string
    ***/
    String digestB64 = new sun.misc.BASE64Encoder().encode(digest);
    return digestB64;
}

PTI digests are stored in a lookup table maintained in the physical memory of the router. A
lookup table has a maximum capacity which may be restricted to as few as 100,000 PTIs.
The amount of router memory available is very limited and there is a constant flow of
packets through the router. As a result of this, PTI digests must be periodically removed from
the lookup table. While there are a number of different policies for replacing items in a list,
our analysis is based on the assumption that any new PTI that would cause a table overflow
replaces the oldest table entry.
Hash-keyed Message Authentication Code

Message authentication codes, or MACs, are commonly used to ensure data integrity and authenticity. “The sender appends to the data D an authentication tag computed as a function of the data and a shared key. At reception, the receiver recomputes the authentication tag on the received message using the shared key, and accepts the data as valid if this value matches the tag attached to the received message.” (Bellare, 1996) “A keyed-hash message authentication code, or HMAC, is a type of message authentication code (MAC) calculated using a cryptographic hash function in combination with a secret key.” (“HMAC”, 2005)

HMACs are used to derive the locks stored in each router lookup table. After the router has calculated the PTI digest for a given packet, the router forms an HMAC from the packet's PTI digest and a secret key known only to that router. This HMAC could be created using either the MD5 or SHA-1 hash functions, both of which seem to be popular in router implementations. In the case of the MD5 hash function, the output is 128 bits long, as per specifications. The 128 bits are divided into two portions: the right and left segments. The left segment, which has a length of \( n \) bits, is used as the lock for the packet and is used as an index into that router's lookup table. The right segment, which is \( 128 - n \) bits, is called the Packet Identification Number (PIN). This PIN, along with the packet's PTI digest, is used to form the packet's stamp. (Please see Figure 2.) During traceback, the PTI digest is used to form the same HMAC as mentioned above. The PIN is then compared against the last \( 128 - n \) bits of the HMAC. If they are identical, the first \( n \) bits are used to “unlock” the router's table and allow traceback to continue.
Figure 2. Packet Stamp Creation. When router $R_i$ receives a packet, it updates a hashing algorithm (i.e. MD5) with the router's secret, the arrival time interval, receiving link MAC address, and the previous-hop router's MAC address. The router produces the PTI$_i$ digest as a result and uses this along with the router's secret to update an HMAC algorithm to produce authentication code. This code is then split into an $n$-bit left segment, or lock$_i$, and the 128-$n$ bit (in the case of MD5) right segment, or PIN$_i$. The router combines the lock$_i$ with the stamp from the previous-hop router to form a row in the lookup table. PIN$_i$ is concatenated on to the PTI$_i$ and embedded into the packet, replacing PTI$_{i-1}$.

**Stamp**

A stamp is changed whenever a non-traceback packet is handled by a router. The initial value given to a stamp is undetermined. That is, any value is possible and could be decided by the sender or any mechanism. In either case, the initial value is unimportant. Call this value stamp$_0$. The value of stamp$_i$ is a key belonging to the $i$th router on the packet's path. The value of stamp$_i$ depends on a quantity specific to the receiving link, previous-hop router, arrival time interval of the packet, and a router-specific secret key. To form the new stamp, the $i$th router forms the PTI digest corresponding to the given packet. This PTI
digest is then used to form the HMAC and the result is split into the left and right segments. The right segment is concatenated on to PTI, and the resulting key is embedded in the packet. (See Figure 3.)

Figure 3. Routing and stamping of IP packets. Router $R_i$ is about to receive the packet $p$ during time interval $t_{i-1}$. The state of the lookup table is shown right. During time interval $t_i$, the packet $p$ arrives at $R_i$, is processed and a new stamp is created. This stamp is embedded in $p$ and an entry is added to or replaced in the lookup table on $R_i$. During time interval $t_{i+1}$ the packet $p$ is forwarded by $R_i$ to the next-hop router along the path towards $p$'s destination.
Traceback

As previously mentioned, traceback begins with the transmission of a traceback packet from the victim. The victim first identifies a spoofed attack packet and extracts the embedded stamp. It forms a traceback packet and embeds this stamp in that packet. The victim also creates an HMAC, using a secret key, the embedded stamp, and the source and destination addresses of the traceback packet. The result is embedded in the checksum field. The traceback packet is then sent to the victim's gateway router and traceback commences.

The format of a traceback packet consists of five fields embedded in the data portion of each packet. (See Figure 4.) These fields are the Stamp field, the MAC field, the AS ID field, the checksum field, and the signature field. The stamp field contains the stamp extracted from the attack packet the victim wishes to trace and is immutable during traceback. The MAC field is initially empty and will contain the MAC address of the true source of the errant packet when traceback ends. The AS ID field is used to identify the asynchronous system the attacking host belongs to and is initially empty. The checksum field is used to store a checksum created by the victim and is designed to prevent modification of the stamp field during traceback. The signature field contains the signature of the last router to handle the traceback packet during the traceback sequence. The IP packet is then modified to signal the presence of a traceback packet. The IP packet's source address is that of the victim and the protocol field is set to a decimal value in the range of 56-60 or 101-254. These value ranges are unassigned by the Internet Assigned Numbers Authority (IANA) and should be available to signal packets belonging to the traceback protocol. (Reynolds, 1994) The receiving address of the IP packet should be set to the address of the victim for consistency and is not used by the routers during traceback.
Figure 4. Format of the traceback packet. Five fields required for traceback are embedded into the data portion of an empty IP packet. These fields allow routers to store traceback results, which are returned after traceback is finished.

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stamp</td>
<td>50 b</td>
<td></td>
</tr>
<tr>
<td>MAC</td>
<td>48 b</td>
<td></td>
</tr>
<tr>
<td>Checksum</td>
<td>128 b</td>
<td></td>
</tr>
<tr>
<td>AS ID Field</td>
<td>64 b</td>
<td></td>
</tr>
<tr>
<td>Signature</td>
<td>1024 b</td>
<td></td>
</tr>
</tbody>
</table>

Upon receiving a traceback packet, the router retrieves stamp from the traceback packet. The router then splits the stamp into the PTI digest and PIN components. Using its secret key and the PTI digest, the router forms the HMAC for the packet. The router then examines the last 128 – \( n \) bits of the HMAC and compares these bits to the PIN retrieved from the traceback packet. If these bits match, the router uses the first \( n \) bits to index into the lookup table. If the router handled the original attack packet corresponding to the
traceback packet, the first \( n \) bits of the HMAC should index properly into the lookup table. (See Figure 5.) If successful, the router confirms with high probability that it handled the attack packet. The router now knows, with high certainty that the errant packet, referred to by the traceback packet, was received on the link, from the router, and during the time interval specified by the embedded PTI. If the router is unable to match any of the components, the errant packet was not originally handled by the current router, or the corresponding time interval is too old and has expired. If it was discovered that the router did not handle the packet, the router ceases to process the traceback and drops the traceback packet.
Figure 5. How a router checks stamp validity. When a router receives a traceback packet, it extracts the embedded stamp and splits it into the PTI and PIN components. The router then creates authentication code by updating an HMAC algorithm with the retrieved PTI and the router's secret. The result is then split into two components: the n-bit lock and the 128-n bit PIN'. PIN/ is compared against PIN, extracted from the stamp. If a match is found, the router attempts to index into the local lookup table.

If there was a successful matching, the router finds a new key at the lookup table entry referenced by the first n bits of the HMAC, or the lock. The router then replaces the current PTI and PIN from the traceback stamp with those found in the referenced lookup table entry.
The following steps occur before the packet is forwarded:

1. The router then fills in the MAC field of the traceback packet with the MAC address of the next router along the traceback path.
2. The AS ID field is filled in with the ID of the AS responsible for the current router.
3. The router then signs both the MAC and AS ID fields and places the signature in the signature field. If these fields are already occupied, the router over-writes the current values.

After these steps are completed, the router forwards a copy of the traceback packet to the next router, along the traceback path, via the link corresponding to the router and link fields of the removed PTI, respectively. This process is repeated at each router as the traceback packet attempts to traverse the exact physical path the attack packet took to arrive at the victim. This traceback process terminates and returns information to the victim in two cases:

1. the next router fails to return results
2. the router identifies the router component of the PTI as matching a host and not a router

Regardless of success or failure, the victim receives the traceback packet containing information concerning the path to its attacker. Upon receiving a traceback packet, the victim first checks the checksum field for tampering. This is done by recreating an HMAC with the stamp field, a secret value, and the source and destination addresses. If the value of the HMAC is identical to the checksum field, the stamp field was not tampered with during traceback. The victim then retrieves the AS ID from the packet, uses the ID to retrieve a certificate for the corresponding AS, and uses the retrieved certificate to verify the signature of the traceback packet. If the signature is valid, the victim now knows the packet really came from a host identified by the MAC address in the MAC field, in the AS identified by the AS ID field.
In cases where the traceback process does not successfully finish, the victim must interpret the results of the traceback process. Depending on how much of the attack packet's true path is identified, the victim might be able to identify whether or not a packet is spoofed by determining if the termination point lies along the path to the supposed spoofed address. Each spoofed packet has a true path P1 with a length L1, and a spoofed path P2 with a length L2. In order for P1 and P2 to be different, they must differ in at least 1 hop. In the case where an attacker is spoofing a host on his/her local network, the last hop (last router to end host) would be different. P1 and P2 can have at most n hops in common, where n < L1 <= L2. If the termination point is beyond the nth point, the victim knows the packet is spoofed if the point is not on the path from the victim to the spoofed address. If the termination point is before or at the nth point, the victim will not be able to determine the true source of the packet.

Traceback can fail in either of two cases. The first case involves extreme link congestion. In cases of bandwidth-exhausting attacks or activities, traceback packets sent to a router could get dropped. This could be mitigated by the use of a connection-oriented protocol like TCP. Failure could also result if the traceback packet is sent after too long of a delay. For example, each router stores n time intervals. A router r receives an attack packet during the ith time interval. Assuming each new time interval record replaces the oldest record, the ith record will be replaced after n time intervals have elapsed. The victim receives the attack packet and initiates a traceback. If the traceback packet reaches router r after n intervals expire, the ith entry is no longer valid for the traceback packet (and attack packet). When the router attempts to index into the lookup table, it fails to find the PTI generated from the traceback stamp. Note that the router will be unable to differentiate between this situation and situations where the packet was not handled by the router.
Resistance To Attacks

There are a number of ways an individual can attempt to interfere with this traceback scheme. These methods include, but are not limited to:

1. artificially create a stamp that provides a false negative (tests negative for spoofing)
2. embed extraneous information into original packet to confuse traceback scheme

False Negatives

Assume an attacker is able to intercept packets between the last router and the victim. He or she could attempt to modify or replace an existing stamp on a spoofed packet in order to trick the traceback system. As previously stated, the stamp contains no information about the information the PTI digests represent, so the attacker will have no indications about what specific combination of PTI digests would result in the traceback system returning a false negative. Also, the user does not know how to calculate the PTI digests, nor does he or she have the individual router secrets necessary for doing so.

In order to return a false negative, the traceback packet must successfully traverse enough routers from the victim to form a false path to the network containing the spoofed address. At each router the stamp must index into the table at a particular entry corresponding to next router along the false path. To be accepted by that particular router, the link specified by the PTI must have received packets during the time interval specified by the PTI digest.

Attackers could approximate the time at which a packet was handled by a particular router by using commands like traceroute. This alone is not enough to subvert the system, because the attacker must then successfully determine the interval during which the packet was received and the corresponding PTI digest. The time interval lengths are assumed to have been published and therefore the attacker could approximate the time interval during which a
packet was handled by a specific router. The attacker does not know any salts or secrets used by routers along the path from victim to the spoofed address. The attacker is not able to subvert the system because the attacker cannot forge packet stamps and PTI digests without these salts and/or secrets.

Attackers must also have extensive knowledge of the networking topography from the victim to the network of the spoofed address. He or she must have prior knowledge of exactly which routers and links the traceback packet would traverse in order to form the path to the spoofed host. In the case the attacker was able to procure all this information, he or she would not be able to subvert the system without the salts and/or secrets used to create the PTI digests.

In the case the attacker intercepts and destroys all traceback packets, the victim could assume the traceback packets correspond to spoofed packets. This act will completely stop any attempts at traceback but would serve as a sign that a malicious entity is either intercepting packets between the victim and the nearest router, or has corrupted the nearest router. Once this issue has been addressed, attempts at traceback can resume.

**Extraneous Information**

Attackers may also choose to embed extraneous information into packets on initial transmission in order to confuse the traceback system. Each router along the path of a packet contributes its PTI digest to the packet stamp, while using an HMAC to maintain data integrity. At no other time does the router interact with the stamp contained in arbitrary packets. An attacker could coordinate his or her attack with other attackers in order to map PTI components to links and time intervals. If such an attacker was able to collect PTI components, he or she could attempt to use replay attacks to induce false negative from the traceback system. This issue is largely negated by refreshing PTI component values over time and by using relatively short time interval lengths.

The time component of each PTI digest is unique to a specific time interval on a
specific router. By this we mean each unique time interval, for a unique router, maps to a
unique PTI time component. The receiving link and previous-hop router values hash to
unique link and router components, but are unique to a set amount of time intervals. For
example, packets are received on a link L from a router R. For all packets received on L
from R during a set of j of n total time intervals, the PTI link and router components are all
the same. Because the link and router components are used as an index into the lookup table
on each router, the number of entries in each router's lookup table is proportional to how
often the router changes the values used for these two components. Any lookup table will
contain more entries if the corresponding router changes the router and link components
often. Replay attacks are prevented by changing the PTI digest components often.

The use of hashing and HMACs helps to promote data integrity, authenticity, and
confidentiality. Data integrity is maintained by the checksum field and the router signature in
the traceback packet. The checksum field contains the output of a one-way HMAC function
which uses a secret specific to the end-system. This field ensures data integrity because the
value stored in this field is the result of a one-way function that only the victim is capable of
recreating. The length of this field is either 128 bits or 160 bits, depending on whether MD5
or SHA-1 is used as the underlying hash function. The signature embedded in the traceback
packet by the router should be secure since the public key infrastructure (PKI) binds each
router to its own specific certificate. This signature also allows the victim to authenticate the
traceback results embedded in the traceback packet. Because hashing and HMACs are one-way
functions, an attacker would not gain any secrets or salts used, if he or she should create a
collision.

Proof Of Correctness

The proposed traceback scheme traces packets by the use of two opposite processes:
stamping and “unstamping”. The first operation occurs while the attack packet is being
forwarded from the source towards the destination. The process of stamping results in an
embedded stamp containing encoded path information in the form of a key. The second process involves incrementally unraveling the real path from the destination back to the true source, using the traceback packet, its embedded stamp and information stored at each router. This “unstamping” process only occurs through routers which were likely to have handled the original errant packet.

The computations involved in both the marking (stamping) and traceback (unstamping) phases are simplistic when viewed from a high level. For any packet there is an associated true path called path\textsubscript{true}, which is the path the packet takes to its destination. Path\textsubscript{true} contains \(n\) routers between its source and destination. The stamp the destination finds in a given packet it receives is known as stamp\textsubscript{n}, or the stamp embedded in the packet after modification by the \(n\)th router. As each packet is processed by a router, the router forms a PTI (Packet Traffic Information) digest and derives an HMAC (hash-keyed message authentication code) from the PTI. It then combines a portion of this code, the PIN, with the PTI and embeds the result in the packet. The PTI/PIN combination from the previous-hop router is stored in the lookup table using the first \(n\) bits of the previously-mentioned HMAC. Thus at the \(i\)th router, for a given arbitrary packet:

\[
\text{stamp}_i = \text{PTI}_i + \text{PIN}_i
\]

where:

- \(\text{PTI}_i\) is the PTI digest formed by the \(i\)th router
- \(\text{PIN}_i\) is the last 128 - \(m\) bits of the HMAC formed the \(i\)th router
- \(\text{stamp}_i\) is the stamp embedded in the packet forwarded by the \(i\)th router
- “+” denotes concatenation, not the mathematical addition operation

When leaving the router \(R_i\), stamp\textsubscript{i-1} is stored in the lookup table and is referenced in the future by \(lock_i\). When the packet is received by the next router, this stamping process repeats
until the packet has reached its destination.

During traceback, the victim embeds the stamp it sees into a traceback packet. This final stamp, or stamp\(_n\), is the stamp embedded in the packet by the last router in an \(n\)-length path, or stamp\(_n\). stamp\(_n\) is:

\[
\text{stamp}_n = \text{PTI}_n + \text{PIN}_n
\]

When router \(R_n\) receives this packet, it extracts stamp\(_n\) and matches this to the lock\(_n\).

HMACs are considered one-to-one functions and as a result, stamp\(_n\) can only unlock lock\(_n\).

When the router maps stamp\(_n\) to lock\(_n\), it retrieves stamp\(_{n-1}\). This stamp is embedded in the traceback packet and the packet is then forwarded to the router identified by PTI\(_n\). This mapping of stamps to locks to stamps continues until the traceback fails or the process reaches its target, the true source of the original attack packet.

According to the above description, PTI\(_n\) + PIN\(_n\) maps directly to lock\(_n\), or:

\[
\text{PTI}_n + \text{PIN}_n \Rightarrow \text{lock}_n
\]

Because lock\(_n\) reveals PTI\(_{n-1}\) + PIN\(_{n-1}\) when used as an index in to the lookup table, it can be said that lock\(_n\) maps directly to PTI\(_{n-1}\) + PIN\(_{n-1}\), or:

\[
\text{lock}_n \Rightarrow \text{PTI}_{n-1} + \text{PIN}_{n-1}
\]

Combining the two above relationships, we realize that PTI\(_n\) + PIN\(_n\) maps indirectly to PTI\(_{n-1}\) + PIN\(_{n-1}\):

\[
\text{PTI}_n + \text{PIN}_n \rightarrow \text{PTI}_{n-1} + \text{PIN}_{n-1}
\]
Because this is a recursive relationship,

\[
\text{PT}_n + \text{PIN}_n \rightarrow \text{PT}_{n-1} + \text{PIN}_{n-1} \rightarrow \text{PT}_{n-2} + \text{PIN}_{n-2}
\]

And:

\[
\text{PT}_n + \text{PIN}_n \rightarrow \text{PT}_{n-1} + \text{PIN}_{n-1} \rightarrow \cdots \rightarrow \text{PT}_1 + \text{PIN}_1 \rightarrow \text{stamp}_0
\]

where \(\text{stamp}_0\) is the initial stamp value embedded by the source.

After the traceback packet has traversed the path of the original errant packet, the packet arrives at the first router in the attack path. This router generates the \(\text{PT}_1\) and matches it to an entry in its lookup table. At this point the router identifies the router component as matching an end-system and not a router. The process terminates and the router returns results to the victim.

**Analysis**

In order for an attack to compromise the PTI, PIN combination, he or she must attempt a brute force attack against the function used to derive the HMACs of a particular router. “In cryptoanalysis, a brute force attack is a method of defeating a cryptographic scheme by trying a large number of possibilities; for example, exhaustively working through all possible keys in order to decrypt a message.” (“Brute Force Attack”, 2005)

In order to justify the number of bits needed for the PTI and PIN fields, information must be gathered about how fast computers can conduct brute force attacks against HMAC functions. To test this I developed a small C++ program, which makes use of the Crypto++ libraries (“Crypto++ Library”, 07/2005). Based on the tests (see Appendix A for source code) run on a 2.4GHz PC, on a Pentium III 866 MHz processor, and on a Celeron 2.20GHz
processor, I am able to estimate a 3.20 GHz Pentium 4 processor (the fastest available on Dell's web site) would be able to perform approximately 30,900,000 combinations over a 5 minute period. The program was run 5 times on each CPU and the results were averaged. In each case, the number of combinations grew roughly proportional to the growth of the processor speed.

The goal of the proposed solution is to have the PTI/PIN combinations expire after 5 minutes, leaving a victim a short period in which to initiate a traceback. In the case the attacker is flooding the victim's connection, the traceback requests may not get through. However, as the attack drops off (gradually or sharply), the victim should be able to get a traceback request through. According to the Wikipedia web page on birthday attacks (http://en.wikipedia.org/wiki/Birthday_attack), we expect the attacker to create a collision after \(1.2 \sqrt{n}\) times, where \(n\) is the maximum number of combinations the HMAC function supports. To prevent the attacker from successfully brute-forcing the HMAC function, we must make sure \(1.2 \sqrt{n}\) is at least greater than the maximum number of combinations the fastest computer can attempt in 5 minutes. If a fast computer is able to do approximately 31,000,000 HMAC combinations over a 5 minute period:

\[
1.2 \sqrt{n} > 31,000,000 \\
\sqrt{n} > 25,833,333 \\
n > (25,833,333)^2 \\
n > 667,361,093,888,889
\]

At this point, the HMAC function must support at least 667,361,093,888,889 combinations. To find out the bits we would need:

\[
n = 2^k \\
\log_2(n) = \log_2(2^k)
\]
\[ \log_2(n) = k \cdot \log_2(2) \]
\[ k = \frac{\log_2(n)}{\log_2(2)} \]
\[ k = \frac{\log_2(n)}{1} \]

\[ k = \log_2(667,361,093,888,889) \]
\[ k = \frac{\log (667,361,093,888,889)}{\log (2)} \]
\[ k = 49.250 \text{ bits} \sim 50 \text{ bits} \]

Note that this is the minimum number of bits needed. I do not have access to any computer faster than 2.4GHz, so I do not know how many combinations a 3.2GHz processor would be able to produce in five minutes. The 50 bits applies to the PTI/PIN number combination. If we divide this into a 20 bit PTI and a 30 bit PIN number, we would need to store approximately 118 bits per entry in the lookup table. (20 bit PTI + 128 HMAC – 30 PIN).

If we had 10MB (approximately 83,886,080 bits) to work with for the lookup table, this would give us approximately 710,898 entries in the table.

**Insufficient Router Memory Available**

In the case evidence comes to light that shows we do not have enough free memory on routers, adjustments must be made. One could consider the following modifications:

1. Mark a portion of packets forwarded by the router.
2. Reduce the size of the lookup table index.
3. Increase the length of time intervals.

**Marking A Portion Of The Packets**

Marking a portion of the packets would not be feasible modification. This is because it creates a situation which is more prone to a denial-of-service attack, it violates the
requirement that we allow the traceback of any individual packet, and it would require a fast and reliable way for the victim to determine if a packet contains a stamp. By marking a percent of the total packets being forwarded, we are reducing the amount of work a malicious entity has to do to cover his/her tracks. Because even a single packet could give away an attacker's true location, he or she must intercept and/or tamper with all stamps in his or her packets. If we mark 5% of the attacker's 1,000,000 packets, the attacker is concerned with 50,000 packets instead of 1,000,000. If the attacker is able to prevent the victim from receiving the 50,000 stamps, the victim has no way to determine the attacker's true location. Any packets the victim receives will not contain a stamp and therefore cannot be used for traceback, a violation of our fourth restriction.

In the case a percent of the forwarded packets are marked, the victim and routers must have an easy way to determine if a packet was marked with a stamp. First, this would require a significant amount more bits to be stored in the packet headers. A naive method would be to append a simple bit to the end of the stamp. This bit would be set to a one value if there was a stamp included in the packet and a zero if there was no stamp included. This method is not secure because any entity can intercept the packet and change the bit. To compensate for this, the routers could embed a signature into the packet along with the stamp, but a digital signature would require too many bits to fit into the available header space.

Reducing The Size Of The Lookup Table Index

To reduce the size of a lookup table's index, one must decrease the output of the HMAC function used. The output of the MD5-HMAC function is 128 bits, as per specifications. The index size of a lookup table is determined by splitting the 128-bit output of the MD5-HMAC function between the n-bit lock and the (128-n)-bit PIN. Excluding the overhead of a hash table for simplification purposes, the size of the jth entry in the lookup table at the ith router is:
\[ |row_{ij}| = |\text{lock}_j| + |\text{PTI}_{i-1}| + |\text{PIN}_j| \]

Because the number of bits used for each PIN is uniform across all routers, we get:

\[ |row_{ij}| = n + p + (128 - n) \]

OR

\[ |row_{ij}| = 128 + p \]

Note that the size in bits of each row is dependent upon the output of the HMAC and the length of the PTI digest. The length of the PTI digest cannot be decreased without increasing an attacker's ability to create collisions and therefore trick the traceback system into returning false results. Because the PTI digest lengths cannot be modified, we are left with one option: decrease the output of the MD5-HMAC function. This option is not viable because decreasing the bit-length would make the traceback process more vulnerable to attempts by the attacker to induce false results. Also, decreasing the size of the index could potentially create situations where the same index maps to more than one value.

**Increasing Time Interval Lengths**

This method has the least negative impact upon our proposed system. Increasing the time interval lengths impacts on the memory required by the router as well as the percent of bandwidth wasted when routers need to transmit several copies of an individual traceback packet (i.e. as a result of a lock mapping to more than one entry in the lookup table). Because of this, we need to determine the trade offs between the memory needed and the bandwidth wasted in terms of the time interval length.
A. Lookup Table Memory

Our first step is to derive a relationship between the memory needed and the length of time intervals. First, we wish to store time intervals for five minutes, or 300,000 milliseconds. Each router could potentially handle packets on multiple interfaces from multiple hosts in a given time interval. Because PTI digests differ depending on the receiving link and the previous-hop router involved, we must include a maximum of \( \frac{300,000}{T} \) combinations per link/previous-hop router combination. Thus:

\[
N = \left( \frac{300,000}{T} \right) \times \left( \sum_{j=1}^{i} PR_j \right)
\]

where:

- \( N \) = number of rows in the lookup table at router \( R_i \)
- \( T \) = length of time intervals in milliseconds
- \( I \) = total number of links available at router \( R_i \)
- \( PR_j \) = number of peer routers reachable from link \( j \)

Because the number of interfaces available includes all disabled interfaces, and the number of peer routers available on a particular link is slow to change over time, we hold the summation constant. Upon reorganization we find that if we hold the summation portion constant, we get:

\[
N = \frac{C}{T}
\]

where \( C = 300,000 \times \sum PR_j \)
The maximum memory needed by a table is the total of the memory used by all rows. Thus:

\[ M_{\text{max}}(R_i, T) = \sum_{j=1}^{N} |\text{row}_j| + \text{overhead of the hash table} \]

For simplicity's sake, we choose to ignore the overhead of using a hash table. The size in bits of \(R_i\)'s jth row, \(\text{row}_j\), is as follows:

\[ |\text{row}_{ij}| = |\text{lock}_j| + |\text{PTI}_{i-1,j}| + |\text{PIN}_{i-1,j}| \]

where:

- \(\text{lock}_j\) = index value that maps to the jth lookup table entry
- \(\text{PTI}_{i-1,j}\) = PTI digest contributed by \(R_i-1\) and stored at the jth entry
- \(\text{PIN}_{i-1,j}\) = PIN value contributed by \(R_i-1\) and stored at the jth entry

If we split 128-bit HMAC output into an m-bit lock and a 128-m bit PIN, we have:

\[ |\text{row}_{ij}| = m + p + (128 - m) \]

where p is the number of bits needed to represent the N lookup table entries

If we have N combinations, it would take \( \log_2(N) \) bits to represent the rows of the table. Thus:

\[ |\text{row}_{ij}| = 128 + \log_2(N) \]

OR

\[ |\text{row}_{ij}| = 128 + \log_2(C/T) \]
Because all entries, indexes and values, are the same in length, we can simplify our memory equation:

\[ M_{\text{max}}(R_i, T) = N \left[ 128 + \log_2 \left( \frac{C}{T} \right) \right] \]

OR

\[ M_{\text{max}}(R_i, T) = \left( \frac{C}{T} \right) \left[ 128 + \log_2 \left( \frac{C}{T} \right) \right] \]

OR

\[ M_{\text{max}}(R_i, T) = \left( 128 \frac{C}{T} \right) + \left[ \frac{C}{T} \log_2 \left( \frac{C}{T} \right) \right] \]

B. Wasted Bandwidth

To determine how the time interval length affects the bandwidth wasted, we first need to know how the time interval length affects the size of a traceback packet. The smallest IP packet, 64 bytes occurs when no data is transmitted and only the IP headers are included. Traceback packets embed five fields in the data portion of the packet. The size of the jth traceback packet, or \( p_j \), in bits is:

\[ |p_j| = |\text{header portion}| + |\text{stamp field}| + |\text{MAC field}| + |\text{checksum field}| + |\text{AS ID field}| + |\text{signature field}| \]
Filling in the field lengths, we have:

\[ |\rho_j| = ( 64 \times 8 ) + |\text{PTI}| + |\text{PIN}| + 48 + 128 + 64 + 1024 \]

Using our previous definitions for PTI and PIN, we have:

\[ |\rho_j| = ( 64 \times 8 ) + ( \log_2 (C / T) + (128 - m) + 48 + 128 + 64 + 1024 \]

Simplifying we get:

\[ |\rho_j| = 1904 + \log_2 (C / T) - m \]

Now, if we send \( M \) different copies of the traceback packet, the bandwidth wasted is:

\[
\text{BW}(T) = \sum_{j=1}^{M} \left( \frac{|\rho_j|}{S_{\rho_j}} \right)
\]

where:

\( S_{\rho_j} \) = the line speed in bits of the interface used to send \( \rho_j \)

Substituting our equation for \( \rho_j \), we get:

\[
\text{BW}(T) = \sum_{j=1}^{M} \left( \frac{[1904 + \log_2 (C / T) - m]}{S_{\rho_j}} \right)
\]
Bibliography


Available from: http://www.anderbergfamily.net/ant/history/.

23 HMAC. Wikipedia; [updated 2005 Mar; cited 2005 Mar]. Available from:

24 Hop, Step, but Don't Jump For Scalable Lower-Layer Forwarding Of Best-Effort Traffic.
Ohta M; [updated 1997 Jul 08; cited 2005 Mar]. Available from:
http://www.isoc.org/inet97/proceedings/F4/F4_1.HTM.


00.txt.


29 Internet structure: Backbones, NAP's and ISP's. Ross K, Kurose J; [updated 2005 Mar;
cited 2005 Mar]. Available from:


32 IP Spoofing LG #63. Sharma K; [updated 2003 Dec 02; cited 2005 Mar]. Available

33 ITPRC - TCP/IP FAQ: What Are The Common TCP/IP Application Protocols?. Oliver
M; [updated 2004 Jul 31; cited 2005 Mar]. Available from:


45 Packet Arrivals On Rate-limited Internet Links. Broido A, Claffy KC, Nemeth E;
Packet Spoofing. Bradley T; [updated 2005 Apr; cited 2005 Apr]. Available from:

Pang R, Yegneswaran V, Barford P, Paxson V, Peterson L. Characteristics Of Internet
Background Radiation. Internet Measurement Conference: Proceedings Of The 4th

Papadimitratos P, Haas Z. Securing The Internet Routing Infrastructure. IEEE
Communications Magazine, 2002; 60-68 p.

Paxson V. An Analysis Of Using Reflectors For Distributed Denial-of-Service Attacks.

Paxson V. End-To-End Routing Behavior In The Internet. IEEE/ACM Transactions On

PC Drive Reaches 500GB. Dahl E; [updated 2005 Apr; cited 2005 Apr]. Available from:

PCQuest : Spying & Spy Catching : IP SPOOFING. Kumar J; [updated 2003 May 05;
cited 2005 Apr]. Available from:
http://www.pcquest.com/content/topstories/spy/103050505.asp.

Products & Services Security Advisories - Cisco Systems. Cisco; [updated 2005 Jun; cited
2005 Jun]. Available from:

Radius Protocol, RFC And Vulnerabilities For This Protocol. Colasoft; [updated 2005
Mar; cited 2005 Mar]. Available from:


RSA Security - 2.1.7 What are Message Authentication Codes?. RSA Laboratories;


Appendix A: Brute-force Attack Test Program Source Code

The following code was used to test brute-force attacks conducted against an HMAC function. The program requires a number of combinations to attempt (given as a command-line argument) and outputs the number of combinations created per second, per minute, and per 5 minutes.

A1) The program requires Crypto++ to be installed prior to compilation. The following source code was compiled under Gentoo Linux, kernel version 2.6.12, using GCC 3.3.5.

```cpp
#include <crypto++/sha.h>
#include <crypto++/hmac.h>
#include <iostream>
#include <sstream>
#include <string>
#include <time.h>
using namespace std;
using namespace CryptoPP;

int main(int argc, const char* argv[]) {
long iterations = 0;
time_t clock_tick = CLOCKS_PER_SEC;
string data = "This is the message we wish to perform the HMAC against."

if(argc < 2) {
    cerr << "Usage: " << argv[0] << " <number-of-iterations> " << endl;
    exit(1);
}

try {
    /**
     * start clock timer
     **/
    double start = clock();

    /**
     * convert data to a byte array
     **/
    char const* cData = data.c_str();

    // ...
byte const* pbData = (byte const*)cData;
unsigned int nDataLen = 64;
byte pbOut[HMAC<SHA>::DIGESTSIZE];

/**
 * Retrieve and store the command line argument
 **/
iterations = strtol(argv[1], NULL, 10);
for(int i = 0; i < iterations; i++) {
    HMAC<SHA> mac;
    const char* cKey;
    ostringstream ostr;
    string key;

    /**
     * create key from counting variable
     **/
    ostr << i;
    key = ostr.str();
    cKey = key.c_str();

    byte const* pbKey = (byte const*) cKey;
    unsigned int nKeyLen = 64;

    /**
     * 1. set the key for the HMAC
     * 2. update the HMAC with the data
     * 3. output the digest from the HMAC
     **/
    mac.SetKey(pbKey, nKeyLen);
    mac.Update(pbData, nDataLen);
    mac.Final(pbOut);
}

/**
 * stop clock and figure out elapsed time, operations per second
 **/
double elapsed = (clock() - start)/clock_tick;
double opsPerSec = iterations / elapsed;

/**
 * Print out results
 **/
cout << "Combinations: " << iterations << endl;
cout << "Time elapsed: " << elapsed << endl;
cout << "Average: " << opsPerSec << " MACs/sec" << endl;
cout << " " << (opsPerSec*60) << " MACs/min" << endl;
cout << " " << (opsPerSec*60*5) << " MACs/5 min" << endl;
}
catch(exception e) {
    cout << e.what() << endl;
    return 1;
}

return 0;
}

#####
# Makefile
#
# Author: Adam Neiman
# Date: 07/25/2005
# Description: Used to compile HMACTest test program under Unix/Linux with GCC
A2) A slight modification was needed to compile the program under Visual C++ version 6.0. I was unable to compile Crypto++ as a DLL (Dynamic Link Library) and was forced to copy all the required C++ header files to the compilation directory. Instead of:

```c
#include <crypto++/sha.h>
#include <crypto++/hmac.h>
```

I had to use:

```c
#include "sha.h"
#include "hmac.h"
```

The following list contains the necessary header files (provided by the Crypto++ package):

1. 3way.h
2. adler32.h
3. aes.h
4. algebra.h
5. algparam.h
6. arc4.h
7. argnames.h
8. asn.h
9. base32.h
10. base64.h
11. basecode.h
12. bench.h
13. blowfish.h
14. blumshub.h
15. camellia.h
16. cast.h
17. cbcmac.h
18. channels.h
19. config.h
20. crc.h
21. cryptlib.h
22. default.h
23. des.h
24. dh.h
25. dh2.h
26. dll.h
27. dmac.h
28. dsa.h
29. ec2n.h
30. eccrypto.h
31. ecp.h
32. elgamal.h
33. eprecomp.h
34. esign.h
35. factory.h
36. files.h
37. filters.h
38. fips140.h
39. fltrimpl.h
40. gf256.h
41. gf2n.h
42. gf2_32.h
43. gfprypt.h
44. gost.h
45. gzip.h
46. haval.h
47. hex.h
48. hmac.h
49. hrtimer.h
50. ida.h
51. idea.h
52. integer.h
53. iterhash.h
54. lubyrack.h
55. luc.h
56. mars.h
57. md2.h
58. md4.h
59. md5.h
60. md5mac.h
61. mdc.h
62. misc.h
63. modarith.h
64. modes.h
65. modexpopc.h
66. mqueue.h
67. mqv.h
68. nbtheory.h
69. network.h
70. nr.h
71. oaep.h
72. oids.h
73. osrng.h
74. panama.h
75. pch.h
76. pkcspad.h
77. polynomi.h
78. pssr.h
79. pubkey.h
80. pwdbased.h
81. queue.h
82. rabin.h
83. randpool.h
84. re2.h
85. re5.h
86. re6.h
87. resource.h
88. rijndael.h
89. ripemd.h
90. rng.h
91. rsa.h
92. rw.h
93. safer.h
94. seal.h
95. secblock.h
96. seckey.h
97. serpent.h
98. sha.h
99. shacal2.h
100. shark.h
101. simple.h
102. skipjack.h
103. smartptr.h
104. socketft.h
105. square.h
106. stdcpp.h
107. strcipp.h
108. tea.h
109. tiger.h
110. trdlocal.h
111. trunhash.h
112. ttmac.h
113. twofish.h
114. validate.h
115. wait.h
116. wake.h
117. whrlpool.h
118. winpipes.h
119. words.h
120. xormac.h
121. xtr.h
Appendix B: Definitions And Terms

1) **asynchronous system** - portion of a network or networks which fall under a single administrative organization
2) **authenticity** - refers to the verification of an entity's credentials
3) **confidentiality** - refers to preventing unauthorized entities access to data
4) **edge router** - router residing on the edge of a network, typically serving as a connector (gateway) between the local network and another network
5) **hash-keyed message authentication code** - a type of message authentication code involving the use of a secret key and a hashing function
6) **hop** - a single step, usually from one router to another, along a network path
7) **integrity** - refers to the general validity of data and lack of modification by an external party
8) **Internet Protocol Security** - a set of protocols designed to support secure transmission of internet protocol packets.
9) **link** - physical connection, commonly a connecting cable or wire, between two or more routers
10) **message digest** - the output of a hashing function, such as MD5.
11) **network address translation** - technique in which the source and/or destination addresses of packets are rewritten when passing through a network router and/or firewall.
12) **next-hop** - next step along a network path
13) **packet spoofing** - refers to a network entity forging the source address of packets
14) **previous-hop** - previous step along a network path

15) **true path** - actual path a packet takes from its true sending host to its destination

16) **true sending host** - the actual host responsible for sending a packet

### Index

1) asynchronous system (AS)  \(22, 26, 39, 77\)

2) attacks
   a) blind attack  \(18\)
   b) Denial-of-service (DoS)  \(3, 9, 16, 20, 21, 24, 66\)
   c) distributed denial of service (DDoS)  \(16, 64\)
   d) man-in-the-middle  \(20\)
   e) packet sniffing  \(8, 20\)
   f) Port scanning  \(20\)
   g) stealth scan  \(9, 20\)
   h) reflector  \(20\)

3) authenticity  \(34, 48, 77\)

4) chargen  \(27\)

5) confidentiality  \(24, 28, 48, 77\)

6) hashing  \(10, 13, 28, 48, 77\)
   a) SHA-1  \(10, 13, 33, 35, 48\)
   b) hash-keyed message authentication code (HMAC)  \(10, 13, 29, 34, 37, 39, 47, 49, 52, 69, 70\)
   c) MD5  \(10, 13, 17, 33, 35, 48, 68, 77\)
   d) message digest  \(10, 17, 24, 32, 77\)

7) ICMP  \(9, 21, 25\)
   a) packet messages  \(9, 22, 25\)
8) Internet Protocol (IP)  
   a) Packet fields  
      i) Destination Address  16  
      ii) Don't Fragment  17  
      iii) Flags  17  
      iv) Fragmentation Offset  17  
      v) Header Checksum  16  
      vi) Identification  17  
      vii) Internet Header Length  16  
      viii) Protocol  16  
      ix) Reserved  17  
      x) Time To Live  16  
      xi) Total Length  16  
      xii) Type of Service  16  
      xiii) Source Address  16  
      xiv) Version  16  

9) Internet Protocol Security (IPSec)  28  

10) key  14, 19, 29, 33, 37, 39, 42, 49  

11) lock  32, 35, 42, 50  

12) media access control (MAC)  33  

13) message authentication code (MAC)  3, 10, 13, 29, 34, 49, 64, 67, 70, 77  

14) Network Address Translation (NAT)  28  

15) Public Key Infrastructure (PKI)  14, 48  

16) spoofing  3, 8, 15, 18, 19, 26, 44, 62, 64, 65, 67, 78  
   a) detecting
i) Active methods 18
ii) Passive methods 19

b) attempts at solving

i) link testing 9, 22, 27
ii) message digest logging 22
iii) pattern filtering 9, 22, 26
   (1) egress filtering 26
   (2) ingress filtering 18, 26
iv) probabilistic packet marking 9, 22, 25

17) stamp 16, 29, 35, 39, 42, 49

a) Packet Traffic Information (PTI) 32, 37, 40, 45, 52, 54
b) Packet Identification Number (PIN) 32, 35, 40, 42, 49, 52, 54

18) Transport Control Protocol (TCP) 9, 15, 19, 21, 44

19) traceback 2, 3, 9, 22, 23, 29, 35, 38, 39, 44, 62, 67

a) packet 29, 39, 43, 49

i) AS ID field 39, 43
ii) Checksum field 39, 43, 48, 60
iii) MAC field 39, 43
iv) Signature field 39, 43
v) Stamp field 39, 43

b) path 31

20) true sending host 15, 29, 78