Identification of Host Audit Data to Detect Attacks on Low-level IP Vulnerabilities

Thomas Daniels, Eugene Spafford
COAST Laboratory[*]
Purdue University
West Lafayette, IN 47907–1398
{daniels,spaf}@cs.purdue.edu
COAST TR 98/10

[*]Portions of this work were supported by contract F30602-96-2-0268 from Rome Labs (USAF), and by sponsors of the COAST Laboratory.

Abstract

Conventional host-based and network-based intrusion and misuse detection systems have concentrated on detecting network-based and internal attacks, but little work has addressed host-based detection of low-level network attacks. A major reason for this is the misuse detection system's dependence on audit data and the absence of low-level network data in audit trails. This work defines low-level IP vulnerabilities and distinguishes between low-level IP and IP-based vulnerabilities. Furthermore, we analyze a number of different low-level IP attacks and the vulnerabilities that they exploit. We develop attack signatures for each attack, and based upon our analysis, we determine a baseline collection of information needed to detect the attacks. We suggest locations within protocol stacks where the needed data can be collected. Finally, we generalize from the baseline audit data to try to predict audit content suitable not only for detecting these attacks, but possible future ones.

1 Introduction

Errors in the design and implementation of low-level network protocols can create vulnerabilities in computer systems. Because it may not be economically feasible or even possible to fix all of these vulnerabilities, effective intrusion detection systems (IDS) are needed that can detect attempts to exploit these vulnerabilities. Additionally, even if a system does not have a particular vulnerability, an IDS can still detect an attack thereby justifying system maintenance and warning that further attacks may be expected. Unfortunately, operating systems do not audit the data necessary to detect low-level network attacks, and network-based IDS have problems that limit their usefulness. This paper specifies audit data that can be used to detect these attacks.

1.1 Low-level Network Protocols

For our purposes, low-level network protocols shall be those protocols that are at or below the Transport layer of the International Organization for Standardization's (ISO) reference model. Further information about the ISO model can be found in [Com95] and [Tan88]. We define low-level IP procotols as those protocols in the TCP/IP Internet Protocol Suite that are low-level network protocols. This includes the Internet Protocol itself and its support protocols such as Internet Control Message Protocol (ICMP), Internet Group Management Procotl (IGMP), and Address Resolution Protocol (ARP), which implement functionality similar to the ISO Network Layer. We also include TCP and UDP, which use the services of the IP layer to provide end-to-end communication similar to that in the ISO Transport Layer. Because IP is implemented over many different data link protocols that use a variety of physical media, we do not consider Data Link or Physical layer protocols as low-level IP protocols.

1.2 Low-level IP Vulnerabilities

In [Krs98], Krsul defines a software vulnerability as "an instance of an error in the specification, development, or configuration of software such that its execution can violate security policy." Similarly, we define a low-level IP vulnerability (LLIV) as an instance of an error in the specification of a low-level IP protocol or the specification, development, or configuration of an implementation of a low-level IP protocol such that its execution can violate security policy. The primary distinction that this definition makes is that there is a difference between the specification of the protocol itself and the specification of a given implementation of the protocol. The distinction is important because it is relatively easy to make changes to one implementation of a protocol while it may not be feasible for all implementations of a protocol to be changed to comply with a modification of the protocol.

Not all vulnerabilities related to IP protocols are low-level network vulnerabilities. For instance, Sendmail vulnerabilities that allow an external attacker to take over the host are caused by errors in implementations of the Simple Mail Transfer Protocol (SMTP). SMTP is implemented using TCP and is therefore at a higher level than TCP and UDP. Similarly, other vulnerabilities in network services that are above the transport layer are not low-level network vulnerabilities. These vulnerabilities in network services above the transport layer are called network-based vulnerabilities.

1.3 Intrusion Detection Systems

Three basic architectures have been proposed for intrusion detection in a networked environment: network-based, host-based, and distributed intrusion detection systems. Network-based intrusion detection systems (NIDS) passively examine raw network traffic at a single point on a network. By analyzing the packets seen on the network, NIDS try to match "signatures" of intrusive behavior or detect anomalous behavior from profiles of network activity. Examples of NIDS are described in [HDL$^+$90] and [Sys]. Host-based intrusion detection systems (HIDS) use system audit trail data and sometimes other interactively determined system information to detect intrusions. As with NIDS, HIDS use various pattern matching algorithms and statistical methods to detect intrusions from collected data. Examples of HIDS are described in [Kum95] and [Ilg92]. Distributed intrusion detection systems take data from various hosts, network components, and network monitors to detect intrusions. These systems may use network data as well as data collection and analysis at multiple hosts. Examples of distributed intrusion detection systems are described in [HJS$^+$93] and [ZBGF$^+$98]

1.3.1 Network-based Intrusion Detection Systems

A NIDS monitors a network by capturing network packets. A NIDS parses the captured packets, analyzes them, and extracts useful information from them. Typically, this is done without changing or inserting any data on the network and is therefore called passive analysis.

To capture packets, a NIDS uses "promiscuous mode" access to the network. In this mode, the NIDS reviews all network traffic that its network interface receives even if the data is not addressed to the NIDS. Figure 1 shows a simple network that includes a NIDS. On broadcast networks like Ethernet, NIDS can monitor a number of hosts simultaneously without affecting network performance or the individual hosts. This allows a NIDS to easily correlate information across different hosts on the network and cause minimal disruption to the hosts on the network.

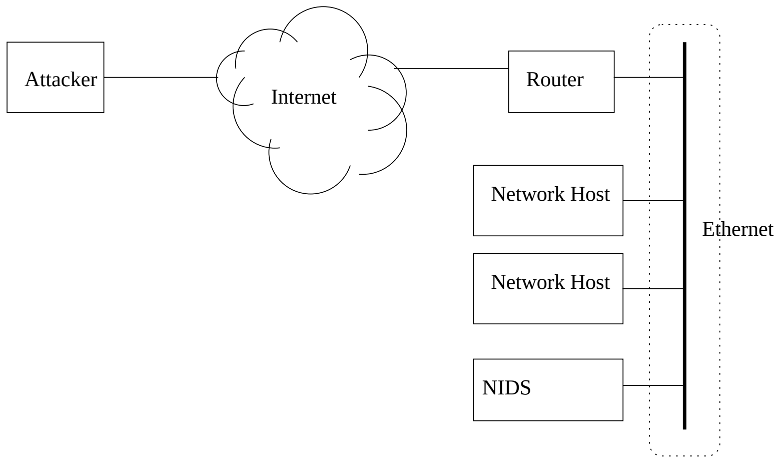

Figure 1. A simple example of a network topology protected by a network-based intrusion detection system. The NIDS sees all traffic on the Ethernet.

At first glance, NIDS seem perfect for detecting attacks on LLNV's, but it has been shown that NIDS that rely solely upon raw network data for their input have fundamental problems. [PN98] These problems include insufficient information available on the network, and vulnerability to denial of service attacks. Information that is not available from passive analysis include network topology such as the number of hops from the NIDS to the host for which a given packet is bound. Details of which operating system is running on a given host and how a particular operating system reacts to unusual network traffic such as overlapping IP fragments are needed by a NIDS to determine how network traffic will affect a given host.

A NIDS must maintain state information about all the hosts that it protects to determine even low-level network behavior. According to [PN98], currently available NIDS do not maintain sufficient state about network traffic to detect many low-level vulnerabilities. For instance, no NIDS surveyed in [PN98] reassembled IP fragments so any higher level traffic contained in a fragmented datagram was ignored. This is probably because the NIDS developers do not wish to maintain enough information about IP fragments to reassemble them for analysis. Additionally, reassembly of overlapping fragments varies between operating systems, which implies that the interpretation of fragments is dependent upon the operating system of their destination. This is information that the NIDS usually does not have. Furthermore, in systems for which source code is unavailable, it may not be easy to determine the precise behavior of an IP implementation.

NIDS have other disadvantages as well. A NIDS presents a single point of failure for intrusion detection efforts. If an attacker can disable a NIDS, all hosts on the local area network (LAN) are then left without an IDS. Furthermore, as NIDS become more sophisticated and try to maintain more state information about network events, they become more susceptible to denial of service through resource starvation. For example, an attacker could send fragments from many different datagrams to many different hosts. The NIDS would have to retain copies of the fragments while waiting for enough of the fragments to arrive so that assembly can take place. By doing this, it is possible that the IDS could either run out of available memory or slow down so much that it misses further network traffic. This might be particularly problematic on networks with small MTU's where fragmentation might be common.

Other aspects of a network may also interfere with a NIDS ability to function. These include switching network hubs, multiple routes to a host, and low-level encryption of network traffic.

Switching hubs are used on Ethernet LAN's to connect multiple hosts in a star topology. Traditional hubs transmitted Ethernet traffic received from one host to all other hosts connected to the hub, but switching hubs send traffic only to the port attached to the recipient host. A NIDS attached to a switching hub would only see traffic that was addressed to it, and therefore it could not detect intrusive behavior for the LAN.

Some networks have multiple routes by which network traffic can be transmitted. For instance, a company that depends on continuous access to the Internet may have two or more independent Internet connections routed to different parts of its internal network. In this case, an attacker might send half of a malicious sequence of packets by way of one route and the remainder of the sequence over a second route. If the NIDS is not on both routes, it may not be able to detect the intrusive behavior.

As more companies begin doing business over the Internet, there is a greater demand for confidentiality of network traffic between organizations, but encryption of network traffic at the network or transport layer denies a NIDS from collecting enough information to detect attacks. In RFC 1826, the Encapsulating Security Payload (ESP) is described. This scheme provides for encryption and encapsulation of IP datagrams to provide confidentiality and integrity of network traffic over IP networks. [Atk95] ESP provides mechanisms that encrypt an IP datagram and encapsulate it in an unencrypted carrier datagram that is then routed to its destination. When the carrier datagram is received, the payload is decrypted and processed by the IP layer of the host. A NIDS would observe the carrier datagram but would have no information about the encrypted datagram that it carries. Because of this, an attacker could send a malformed packet using this mechanism and a NIDS could not detect it.

1.3.2 Host-based Intrusion Detection Systems

In contrast to network-based systems, HIDS reside upon individual computers and primarily use data from operating system audit trails as their input. Some HIDS also make interactive queries of the host system to compensate for data that does not exist in the logs. Because operating system audit trails are geared towards system calls and application level auditing, most HIDS try to detect system misuse and network-based intrusions while ignoring low-level network behavior. In [Pri97], Price showed that even the most detailed auditing systems contained insufficient content for current host-based misuse detection systems.

Among the five operating system audit trails surveyed in [Pri97], only the Basic Security Module (BSM) of the Solaris operating system from Sun Microsystems included any form of low-level network auditing. The BSM documentation [Sun95] states that BSM supports logging of IP datagram headers, but it does not include IP options. Furthermore, [Sun95] does not state whether the headers include those of IP fragments or only those of assembled ones. Are these headers of datagrams that have valid checksums? Even if the BSM IP header logging is satisfactory for detection of IP layer attacks, transport layer data is not audited, which makes it impossible to detect some TCP vulnerabilities from the audit trail.

The other operating systems that Price surveyed were HP-UX, UNICOS, OpenVMS VAX, and Windows NT. These systems, as well as BSM, focus on application level audits and the auditing of system calls. Further information about their audit trails is available in [Pri97, Hew95, Dig96, Cra95, Mic95].

1.3.3 Distributed Intrusion Detection Systems

Distributed approaches to intrusion detection take varying approaches to combining elements of HIDS and NIDS to detect intrusive behavior on systems. Depending on how distributed systems incorporate the features of NIDS and HIDS, they may take on attributes of either HIDS or NIDS.

1.4 Problem Statement

Conventional operating systems lack sufficient audit data to allow host-based intrusion detection systems to detect low-level network attacks. On the other hand, Network-based systems because of their dependence on network traffic are unable to determine enough information from passive analysis to detect all information necessary to detect all intrusions. We propose to determine the audit data that could be added to conventional operating system audit trails to support host-based detection of attacks on low-level network vulnerabilities.

Because of the problems with NIDS described in Section 1.3.1, we choose to look at a host-based approach to detecting attacks on low-level network vulnerabilities. One advantage that HIDS have over NIDS for detecting network attacks is that issues of network topology and packet acceptance are resolved. Data detected by a host has obviously reached it and by choosing at what point in the implementation we audit network information we overcome the problem of knowing whether a host will accept a packet. Furthermore to properly implement the protocols, an individual host must already maintain the state information that a NIDS has to simulate. For instance, an implementation of TCP (by definition) must reorder segments that arrive out of order.

The major problem with using a host-based approach to detect low-level network vulnerabilities is that the needed data is not readily available. As stated earlier, conventional systems do not log network information, and the network information that can be extracted is usually straight from the network interface. Using this raw information would still leave us the problem of having to simulate the protocol stack to use the data.

To solve this problem, we determine audit content requirements to detect low-level IP attacks and then suggest additional audit content for operating system audit trails. To determine the data required, we look at seven different low-level attacks and signatures that can be used to detect them. Additionally, we look at the vulnerabilities exploited by the attacks to find other audit content that might assist a HIDS. From the data requirements implied by each vulnerability, we create a baseline list of audit items and the points at which they should be collected during processing of network data. Finally, we generalize from the list to determine what audit content could be collected to help detect current and future low-level IP vulnerabilities.

2 IP Vulnerabilities and Attacks on Them

This section surveys seven IP vulnerabilities and the methods used to exploit them. Four are errors in the IP layer, and two are errors in the TCP layer. The remaining one is an error where all parts of the system are acting according to their specifications, but the capabilities of the system conflict with expected security policy.

For each vulnerability, an attack signature is developed. This signature is a somewhat informal expression that evaluates to true when an attack attempt is detected. It is important to note that these signatures are for a single host to detect an attack on itself. Additionally, the signatures are not based on any specific audit source, but assume that any information on the host can be made available.

In addition to signature analysis, each vulnerability is analyzed to determine the part of the code, specification, design, or configuration from which the vulnerability originates. As part of this analysis, we look for system behaviors that could be logged to help detect an attack. Again, we assume no preexisting audit source.

Finally, audit requirements are specified for each vulnerability. We first specify the audit data necessary to evaluate the signature and therefore detect the attack. Additionally, we specify other forensic data that may be useful for determining the source of the attack.

The vulnerabilities were chosen according to a number of criteria. First, the vulnerability must exist in the IP protocol suite at or below the TCP/UDP layer. This disqualifies network-based vulnerabilities that occur at higher layers such as sendmail vulnerabilities. Secondly, there must be enough information freely available to analyze and determine a signature for a vulnerability. For instance, low-level IP vulnerabilities exist in Windows NT, but without source code, extensive vulnerability analysis was not possible. Third, vulnerabilities that were not known at the beginning of the research period (February 1, 1998) were not considered. Also, other vulnerabilities that have not appeared in a public forum were not considered.

2.1 ARP Cache Poisoning

In this section we will discuss a vulnerability in the Address Resolution Protocol (ARP). The vulnerability, described in [Vol97], is significant because it allows a machine on a Local Area Network (LAN) to masquerade as other local machines by "poisoning" a target's ARP cache. By doing this, an attacker may be able to exploit trust relationships such as those formed using UNIX's `/etc/hosts.equiv` to gain unauthorized access to other systems on the network. Additionally, an attacker can prevent other computers from using IP-based network services. The vulnerability can not easily be corrected because it is an error in the design of the protocol and not in its implementation.

2.1.1 The Address Resolution Protocol

Defined in RFC 826 [Plu82], ARP provides a mechanism for mapping higher-level network addresses to hardware network addresses. The protocol enables hosts to use IP on Ethernet networks by mapping IP addresses to their corresponding Ethernet addresses.

First, we examine the ARP request portion of the address resolution process. If a machine, call it A, needs to send some IP network traffic to another machine, B, via the LAN, A first searches its ARP cache for an entry with B's IP address. If the search is successful, A uses the cached hardware address. Otherwise, A creates an Ethernet broadcast frame with the Ethernet frame type set to the code representing ARP traffic. A places an ARP message in the frame. The message specifies that it is an ARP request and also the hardware and protocol types for which A is requesting addresses. A then assigns its hardware and high level addresses to the sender address fields of the message. Finally, A assigns B's IP address to the target IP field, and broadcasts the frame.

Because the ARP request is broadcast, all hosts on the LAN receive the ARP request and must respond if the TARGET IP matches their IP address. To respond, B changes the message to an ARP reply and writes B's hardware address into target hardware address field. B then swaps the hardware-IP address pairs so that the sender addresse become the target addresses, and vice versa. The resulting message is put into an Ethernet frame with an ARP frame type and send back to A.

When A receives the ARP reply from B, A notes the sender address fields and can now use the sender's hardware address field to transmit IP traffic to B. It is important to note that implementations of ARP are stateless. Because of this, any well formatted ARP reply received by a machine will be honored even if the machine never transmitted an ARP request. This allows anyone who is capable of sending ARP replies on a LAN to corrupt a host's IP to hardware address mappings.

Earlier we explained that when A wanted to send an IP packet, it would broadcast an ARP request and wait for an ARP reply. If A did this every time it needed to send IP traffic, why not just broadcast the data packet and forego the ARP mechanism? If this were the case, every machine on the LAN would have to examine every frame containing an IP packet to determine if it was the recipient. To avoid the overhead involved in such a scheme, ARP implementations cache the results of recent ARP replies. In addition, when a machine receives an ARP request, the ARP module caches the sender's address information before replying. This eliminates the need for another ARP request and reply sequence if the communication is bidirectional.

Because IP to hardware address mappings are not permanent, the ARP module eliminates or overwrite entries after a timeout period. Address mappings can change when a computer's IP address is changed or when a network interface card is changed in an existing machine. If cache entries did not timeout when unused, the cache would need to be manually cleared when a machine's address changed. Additionally, a computer can limit the amount of address mapping information it maintains because it can use ARP to determine an unknown address – a full list of address mappings is unnecessary.

2.1.2 The ARP Cache Poisoning Attack

The attacks described below rely on the attacker being able to inject false ARP replies onto the LAN. To do this, an attacker must have administrative access to some machine on the LAN. Also, because ARP packets do not cross routers, the attacker must be physically near enough to be on the same LAN.

The first attack variant that we describe causes a denial of service on a host on the network, call it A. The service denied to A is the ability to send IP traffic to another host on the LAN, call it B. The first step of the attack is to send an ARP reply to A that misstates B's hardware address. More specifically, send an ARP reply frame with the target addresses set correctly for A, the source IP address set to B's IP address, and the source hardware address set to a hardware address that does not exist on the LAN. The attack causes A to add or modify B's entry in A's ARP cache so that B's IP address now maps to an incorrect hardware address. When A needs to send IP packets to B, the packets are sent to the wrong hardware address, and consequently, B never sees them.

To maintain the denial of service, the attacker can continue to send fake ARP replies to A. Otherwise, B's entry in A's ARP cache will timeout and cause A to send a new ARP request. Of course, B will respond to the new request with B's proper hardware address. Alternatively, the attacker could monitor the Ethernet waiting for an ARP request and then send incorrect replies after a waiting for B's reply to be sent.

In addition to denying service to a host, ARP replies can be used to gain unauthorized access to computer systems by exploiting trust relationships between systems. To assist with the explanation of this attack, imagine two UNIX computers on the same LAN named A and B. A's `/etc/hosts.equiv` file contains B so any user on B can use `rlogin` to remotely login to A. In other words, A trusts B's users.

The attacker's goal is to make his or her machine, M, appear to be B so that he can log in to A. The attacker can use ARP to do this by sending an unsolicited ARP reply to A. The attacker first disables ARP on M so that he controls all ARP traffic from M. For the attack to proceed, M's IP address should now be changed to B's IP address. Because ARP on M is disabled, machines that are not directly communicating with M will not notice this. The attacker then sends an ARP reply to A with source IP address field set to B's IP address and the source hardware address field set to M's hardware address. A's ARP cache will now map B's IP address to M's hardware address and all IP traffic that A sends to B will be delivered to M instead.

Now, our attacker can `rlogin A` from M to gain access to A. This is because to A's IP layer, it appears that IP traffic is being received from and sent to B. Because the authentication is only based upon the IP address and M is falsifying its IP address successfully, users on M are given access.

2.1.3 Signatures to Detect ARP Cache Poisoning

The attacks described above rely upon an attacker sending a falsified ARP reply frame over the LAN segment. The signature of the attack is simple:

> Does there exist an ARP reply with a Sender IP address, X, and a Sender Hardware address, Y, such that Y is not the correct hardware address for the host with IP address X?

Determining the correct hardware address in this case may be non-trivial because a network's IP to Hardware address mappings are not static. Systems that change IP addresses need not change their network interface hardware or address—indeed, that is one reason that ARP is useful. So address mappings that are changed as part of system maintenance, must be somehow taken into account when scanning an audit trail for this signature. Furthermore, maintaining a network-wide address resolution table defeats the purpose of using ARP to resolve addresses.

Another possible signature could take time into account and look for changes in address mappings. This signature is:

> At time T, an ARP reply is logged with sender IP address, X, and sender hardware address, Y. If at time U, U>T, an ARP reply is logged with sender IP address, X, and sender HW address, Z, such that $Z \neq Y$, then true.

This signature has the same problems with false positives, but it does not require an authority that dictates the correct address mappings. If a machine's address mapping changes legitimately, this signature will match it as a false positive. If B never sends a legitimate ARP reply to A, then the audit trail will not reflect a change in address mappings. This would result in the system missing the attack.

2.1.4 Analysis of ARP Cache Poisoning

ARP cache poisoning is not an implementation problem. The problem is in the specification of the protocol itself. In other words, the problem is not a software flaw, but it is flaw in the specifications laid down in RFC 826. [Plu82] Because the RFC states that ARP replies will be honored regardless of whether the machine made an ARP request and without any form of authentication, the machine should trust that its ARP cache entries are correct only so much as it trusts the integrity of all of the other systems on the LAN.

Even though ARP cache modification attacks require access to the LAN, it is still a serious vulnerability because ARP corruption can be used to gain further access after an initial intrusion. Additionally, problems with ARP may appear benign because administrative access to a machine on an internal LAN is required to launch an attack. However, on machines with simple or no security models such as those running Windows 95 or Mac OS, an unprivileged insider can launch the attack because he or she effectively has administrative privileges on the machine. Furthermore, once the root account of an internal machine has been compromised by an external attacker, ARP related attacks can be used against other machines on the LAN.

2.1.5 ARP Audit Requirements

To evaluate the signatures defined here, the following pieces of information are necessary. Minimally, we must have the sender IP addresse and sender hardware address field values from every valid ARP reply packet received by the host. We must also have some way of ordering the values temporally.

Other identifying information such as the network interface and network from which the ARP message came would be useful for tracking the source of the attack.

2.2 IP Fragmentation

In this section we describe a vulnerability that exists in the specification of the fragment reassembly functionality of the IP layer. The vulnerability, discussed in [ZRT95] and analyzed in [KST98], allows an attacker to defeat firewall packet filtering mechanisms by sending overlapping IP fragments. The section begins with a description of IP fragmentation and some other aspects of the reassembly process as specified in RFC 791. [JP81]. We then describe how overlapping fragments can be used to defeat packet filters, and present signatures to detect this activity. The section concludes with some further analysis of the vulnerability and specification of audit requirements.

2.2.1 IP fragmentation and reassembly

IP datagrams can be as large as 64K bytes, but datagrams must be sent across physical networks whose maximum transmission unit (MTU) is much smaller. To overcome this difficulty, IP implementations fragment datagrams that are larger than a physical network's MTU into smaller datagrams. When the fragments of a datagram are received, they are reassembled using information from the IP header.

The IP header has four fields used specifically for creating and reassembling datagrams.

- Identification: The value of this field is copied to the datagram's fragments if fragmentation is necessary. This field is used in conjunction with protocol number and destination and source IP addresses to identify all fragments corresponding to a single datagram.
- More Fragments (MF) flag: This flag is set if the fragment is not the last one in the datagram. In other words, MF is set if the fragment does not have the greatest offset value.
- Length: The total length of this packet including the header. In a fragment, the length field denotes the length of the fragment–not the length of the assembled datagram
- Fragment Offset: The fragment offset specifies where the fragment data should begin in relation to the beginning of the datagram. It is specified in multiples of eight bytes.

The Fragment Offset field determines where in the assembled datagram the fragments data are placed. It is a 13 bit value that indicates at which multiple of 8 bytes the fragment data begins. The header length field, a 4 bit value, indicates the number of 32 bit words that make up the the fragment header. The Length field, 16 bits, indicates the length of the fragment including the header. The MF (More Fragments) bit is set in all fragments except the one with the greatest offset.

RFC 791 [JP81] presents an algorithm for reassembling fragments into IP datagrams based upon the values of the fields listed above. The given algorithm assembles the data portion of each fragment into the correct area of a buffer based upon the fragment offset and length fields. If two fragments have offsets and lengths such that their data overlaps, the algorithm overwrites the overlapped region with the data from the second fragment (the fragment with the greater offset). RFC 791 does not require that implementations handle overlaps in this way, but it does allow it. The RFC apparently assumes that if two fragments overlap then the overlapping data will be identical.

2.2.2 An IP Fragmentation Attack

Because later fragments can overwrite data from previous fragments, header or other "sensitive" data of higher level protocols can be overwritten by later fragments. For instance, an IP datagram that contains a TCP packet can be fragmented so that the fragments overlap in the TCP header. If the second fragment was constructed so that the data area corresponding to the TCP code bits was different than that in the first fragment, IP implementations that follow the algorithm proposed in the RFC could assemble a datagram that represents a connection request even though the original fragment (offset = 0) did not indicate a connection request.

To see how this detail in the specification can lead to a vulnerability, we look at a network of UNIX systems connected to the Internet via a packet filtering firewall. The firewall has been configured to filter out TCP connection requests from the Internet, but outbound traffic is unrestricted. Because fragments of a datagram can arrive in any order, an attacker can send the second fragment of a TCP SYN packet into the network first. The first fragment of the packet can now be modified so its SYN flag is not set and then sent into the network. When the host receives these fragments they are assembled into a valid IP datagram that contains a TCP connection request. To defeat this attack, the firewall could simply filter out fragments that overlap the sensitive areas of TCP. This suggestion is given and justified in [ZRT95]. However, this does not solve the more general problem associated with other arbitrary protocols that may be transported by IP. If a protocol that is transported in an IP datagram has sensitive fields that lie in an area where fragmentation is likely, the proposed solution would interfere with use of the protocol.

2.2.3 A Signature for the IP Fragmentation Attack

This attack uses fragments that overlap during reassembly and whose overlapping areas contain different data. Because the impact of the attack depends on the higher level protocol being carried in the datagram and the location of overlap, this signature is only based on the fragments and the places where they overlap. The signature is:

> From the sorted list of valid fragments that assemble to make a completed datagram, if two consecutive fragments overlap and that overlapping area differs between the two fragments, then true.

Because we assume that these fragments have had their checksum checked and are well formed, we need not worry about too many false positives caused by transmission errors or defective equipment.

2.2.4 Analysis of the IP Fragmentation Vulnerability

The IP Fragmentation vulnerability arises from attempts to filter network data based on header information while maintaining little state. The vulnerability actually exists in the IP filtering code of the firewall or router that is trying to protect the network. A filter could keep enough information about the fragments it observes to detect an overlapping fragment and eliminate one of the fragments and thereby prevent fragment reassembly. This would effectively prevent the attack from succeeding because IP implementations require all "holes" in the datagram to be filled before reassembly occurs. The most likely reason that filters do not do this is performance and simplicity. Because packet filters may need to examine every packet entering a network, the filter must spend little time for each packet or risk slowing network service. Similarly, a simple stateless filter is much easier to implement than a system where network packets are stored and compared to others as they arrive. The filtering mechanism is further complicated by the fact that IP datagrams and hence fragments can arrive in any order.

2.2.5 Audit Requirements

To support the signature, we could audit all valid pre-assembly IP datagrams, but that would unnecesarily audit all IP data payloads and does not take advantage of information that is probably available on the system. We suggest that since an IP implementation most likely detects overlapping fragments during assembly, the host's implementation could audit only when fragments overlap and include the data from the fragments that are being assembled. This may require direct modifications to the host's IP fragment reassembly module but would prevent audit of all IP data payloads.

Other information that may be useful for tracing the source of the attack would include the network interface from which the fragments were received.

2.3 Teardrop

In this section we present a vulnerability, called Teardrop, in the fragment reassembly portion of the Linux IP implementation. Th vulnerability, described in [Arc97] and [Cen97], allows a remote attacker to crash a Linux system by sending two specially constructed IP fragments. First, this section describes the attack and develops a signature that more precisely describes it. Next, we analyze Teardrop by examining the Linux source code to determine what causes the vulnerability. We conclude by specifying audit content to support detection of the attack.

2.3.1 Teardrop Attack Description

Teardrop attacks send at least two fragments of an IP datagram to the target host, T. To exploit the vulnerability, two of the fragments, call them f_1 and f_2, must be created so that if assembled, one of their data areas would lie within the other's data area. More precisely, assume without loss of generality that f_1.offset < f_2.offset. To exploit Teardrop, the fragments must be from the same datagram, and the following must hold: f_2.offset+f_2.length < f_1.offset+f_1.length. These IP fragments may seem strange, but according to the IP specification, the implementation may overwrite fragments that are overlapped by other fragments, and these fragments are therefor acceptable.

2.3.2 Teardrop Attack Signature

The signature of the Teardrop attack is based upon the sorted list of fragments observed that when assembled form a specific IP datagram. The list is sorted by ascending fragment offset. I call this list F and note that the elements of F have the same protocol fields, identification fields, source addresses, and destination address, and the fragments are observed temporally close to each other. The signature is

> Does there exist f_i in F such that (f_i.offset $\neq$ f_{i+1}.offset and f_i.length $\neq$ f_{i+1}.length) and f_i.offset+f_i.length > f_{i+1}.offset + f_{i+1}.length."

Because F is ordered by offset, the signature will be matched only if fragment f_{i+1} would be assembled inside f_i.

2.3.3 An Analysis of the Teardrop Vulnerability

Teardrop results from a coding error in the Linux IP fragment reassembly code. The `ip_glue()` function from `ip_fragment.c` of the Linux IP module maintains a variable called `offset` that indicates the position in the assembly buffer where the next block of data will be copied. When a fragment overlaps a preceding one, `offset` is modified so that no overlapping data is copied. Another variable, `end`, is set to the byte position in the assembly buffer where the fragment will end. `End` is calculated for each fragment without regard to overlap. Because `end` is not adjusted when a fragment partially overlaps, it will no longer be consistent with the value of `offset`. Furthermore, when a fragment falls completely within another `end` will be less than `offset` so that `end - offset` will be negative. This difference is converted to an `unsigned int` and passed as the third argument to the `memcpy()` function. As a result, `memcpy()` tries to copy a very large number of bytes to a kernel buffer that overflows and crashes the system. Figure 2 shows the normal case where two fragments' data areas only partially overlap in the assembly buffer. Figure 2 shows two fragments that partially overlap. Figure 3 shows an example Teardrop exploit.

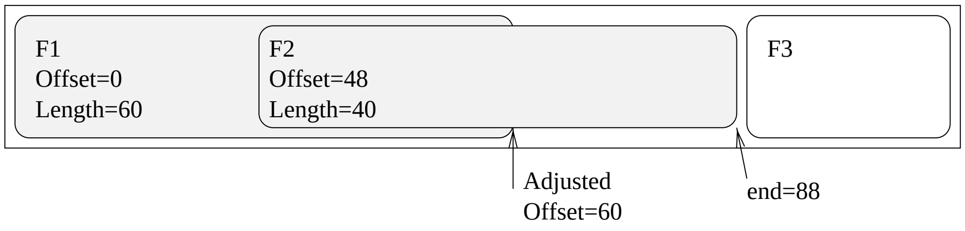

Figure 2. The data payloads of two fragments, F1 and F2, partially overlap in the reassembly buffer. Note that `end - adjusted offset > 0`**.**

2.3.4 Audit Requirements

Audit requirements dictated by the signature are all from valid IP fragments. We need the fragment identifying information so that we can group fragments from one datagram. These data are source and destination addresses, the identification field,

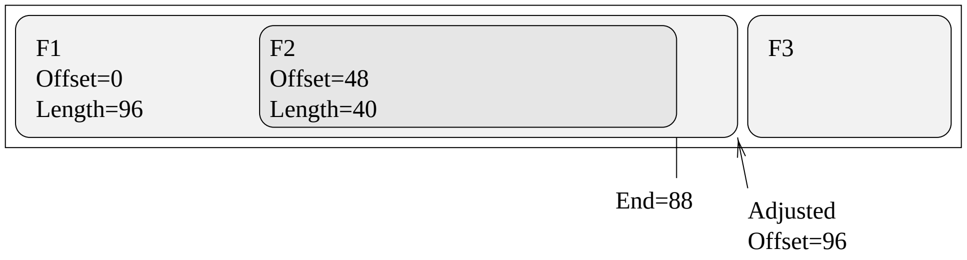

Figure 3. The data payload of the fragment F2 falls completely within the bound of F1. Now, `end - adjusted offset < 0`**.**

and the protocol field. In order to evaluate the signature, we also need fragment offset, datagram length, and the value of the MF bit for each of the fragments.

2.4 Fragment Overrun

This section presents a vulnerability described in [Ken] and [Cen96a] that is commonly known as the "Ping of Death." Actually, the "Ping of Death" is only trivially related to ICMP echo messages, and for this discussion it will be called Fragment Overrun. Fragment Overrun is a vulnerability in the IP fragment reassembly system of several operating systems. When exploited, the vulnerability causes systems to crash or reboot thereby causing a denial of service. The vulnerability is often associated with ICMP because the `ping` command on several different platforms facilitated the attack.

This section first introduces the Fragment Overrun attack. Next, we develop a signature that matches the attack. Also, we perform an analysis of the vulnerability on one of the affected platforms. The section concludes with a statement of audit requirements to detect Fragment Overrun attacks.

2.4.1 Description of the Fragment Overrun Attack

As stated earlier, Fragment Overrun was misnamed because some versions of the `ping` utility facilitated exploitation of this vulnerability. `Ping` is a utility commonly used for testing IP based networks using ICMP echo messages. We will first describe the more general Fragment Overrun attack and then look at why `ping` is useful for exploiting the vulnerability.

The Fragment Overrun attack involves an attacker sending a sequence of specially constructed fragments of an IP datagram to a target machine. For a discussion of IP fragmentation, see section 2.2.1 or [Com95].

A Fragment Overrun attack sends a stream of fragments that if assembled would create an IP datagram of length greater than 65535. To see that this is possible, consider a fragment with offset of 8191, header length of 5 (the minimum), and total length of 65535. When assembled, the fragment would begin at 8*8191=65528 and would occupy the next 65535-5*4 = 65515 bytes. The resulting datagram would be 131,042 bytes long—nearly twice the size of a valid datagram.

To see why this vulnerability has been widely misunderstood as a problem with ICMP, we now look at some features of the `ping` utility. Some versions of ping allow specifying a packet length that is greater than the maximum IP packet length of 65535 bytes. As network MTU's are usually smaller than 65535 bytes, the ICMP echo request packet is fragmented into many IP datagrams that have offset and length fields that represent the invalid size. In other words, the fragments can be created by simply specifying a length greater than 65535 on the `ping` command line. Because these implementations of `ping` are widespread, it is easy to mount this attack.

2.4.2 Fragment Overrun Attack Signature

A simplistic signature for this attack could look at the header of each fragment and determine if that fragment would assemble to a position beyond 65535 bytes. This signature is:

for each otherwise valid IP fragment, (offset*8+total length- header length) > 65535.

This signature is based upon one fragment, but on some systems the reassembly process does not begin until a fragment with the MF bit reset is received and all “holes” in the assembled datagram have been covered. In this case, the signature should also require that all fragments necessary to assemble the datagram have been received and that each will be accepted by the IP layer for reassembly. Because some systems may try to assemble fragments into fixed buffers as they are received, the original and more general signature should be favored.

2.4.3 Analysis of the Fragment Overrun Vulnerability

The vulnerability lies in the IP layer of vulnerable systems. Because of the size limitation on IP datagrams imposed by the IP standard, it is required that IP implementations reject oversized datagrams. The reason that this attack makes systems crash probably varies among the vulnerable systems. It is reasonable to assume that fixed size buffers are being overrun in the kernel thereby corrupting data or code. It is also reasonable to assume that some implementations use unsigned 16 bit variables to maintain datagram lengths and associated variables. When values greater than 65535 are assigned to them, the variables would become inconsistent with the data and could cause erratic behavior.

2.4.4 Audit Requirements

To evaluate the given signature we need the fragment offset, total length, and header length for each valid, pre-assembly IP datagram.

2.5 SYN Flood

In this section, we discuss a vulnerability in the design of many TCP implementations. The vulnerability, described in [Cen96b], allows an attacker to prevent a target computer from accepting new TCP connections thereby denying service to any computer that uses the target’s TCP-based services. The attacker causes the denial of service by sending many TCP connection requests with spoofed addresses to the target machine. Each of these connection requests causes the target to allocate some resource from a fixed size pool. By sending many requests, these resources can be exhausted so that new connection attempts are discarded, and consequently any further access to the machine via TCP are ignored.

In the remainder of this section, we will describe the TCP connection establishment protocol and see why the vulnerability exists. We will show that the vulnerability exists because of design choices made by the designers of TCP implementations.

2.5.1 Description of the TCP Three-way Handshake

TCP provides a reliable, connection-oriented data delivery service based on IP packets. Because it is connection-oriented, TCP must provide for a mechanism to establish a virtual connection between two hosts that wish to communicate. A connection is initialized using a three-way handshake process. Repeatedly, this attack partially completes this handshake until a system can no longer accept new connection attempts. Additionally, since TCP is based upon IP and the attack is nott interactive, the attacker can forge the return address of the IP packets to hide IP address. Such forgery makes tracing the attacker very difficult.

To facilitate description of the connection establishment process, imagine that we have a source host A that needs to create a TCP connection with a destination T. The first step of the three-way handshake is a SYN packet sent from A to T with sequence number X. (Actually a SYN packet is a TCP control packet with the SYN bit set. Similarly, for reset (RST) and acknowledge (ACK) packets. Note also that TCP packets have fields for a sequence number and an ACK sequence number.) When D receives the SYN packet, it replies with a SYN-ACK packet with a new sequence number, Y, and an ACK sequence number of X+1. After S receives the SYN-ACK packet, it replies with an ACK packet with Y for a sequence number. A connection is now established and both sides have established sequence numbers for the following TCP transmission. This process is elucidated in figure 4.

2.5.2 Description of the SYN Flood Attack

The attack, which has become known as a "SYN flood," occurs when many SYN packets are sent to a listening port on a target machine, D. To make them difficult to trace, the SYN packets contain spoofed source IP addresses. Each SYN packet

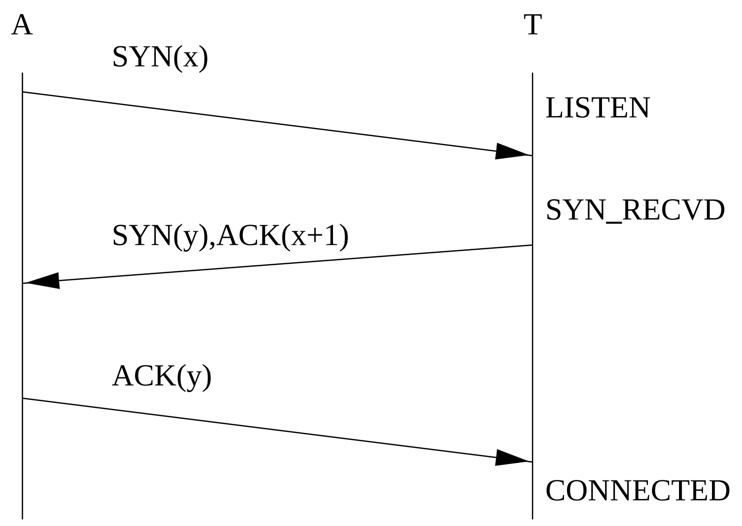

Figure 4. The TCP three-way handshake for connection establishment

sent to D causes D to allocate some resources from a limited size pool. When these resources are exhausted, D ignores any further connection attempts until the timers expire on the half-open connections. When a timer expires, D will reset the connection attempt and free the associated resources. If the attacker wishes to continue the attack, he only needs to continue sending SYN packets to D.

To maximize the effectiveness of the attack, the spoofed SYN packet's source address field should contain an address not reachable by D. This is because D will reply to the SYN packet with a SYN-ACK sent to the source address. If the address is reachable by D, the computer with that address will reply to the SYN-ACK with a RST because it did not expect a SYN-ACK from D. The RST would then reset the half-open connection and free the associated resources. Of course, the attacker need only choose an unassigned address to prevent this.

2.5.3 Detecting SYN floods

To detect a SYN flood from valid TCP packets, an IDS could maintain a list of the ACK packets expected for each SYN-ACK sent by a protected host. By simulating the protocol stack's reaction to the ACK's, RST's, and other network traffic, an IDS could simulate the number of available half-open connections at any point in an audit trail.

Because we are considering a host-based approach, this signature is not really appropriate. It requires maintaining the state of all half-open connections and their timeouts. This duplicates part of the effort of the host's TCP module. An alternative approach would be to have the TCP implementation issue audit events on TCP state transitions. An IDS could then maintain a count of connections in the SYN_RECVD state and compare the results to a threshold to detect a flood.

Of course, the system can issue an audit event when its resources for half-open connections are exhausted. This would make the audit event itself an indicator of a SYN flood. This is easy because a correct implementation must make this check anyway when allocating resources and could create an audit record when the allocation fails. While this simplistic approach would make detection easy, it does not facilitate detection of other kinds of attacks nor does it help determine the source of the flood. The signature described here is

Does there exist an “out of half-open connections” audit event?

2.5.4 Analysis of the SYN Flood Vulnerability

TCP is based upon IP, a best-effort, connectionless protocol. Because of this, when D receives the SYN packet, it must keep track of not only X, but also the timeout information for retransmission of the SYN-ACK packet if it is lost, TCP state information, IP addresses, port numbers, and other information. According to [SKK$^+$97], data structures needed for one half-open TCP connection can exceed 280 bytes. The amount of information needed per connection will vary between implementations of TCP/IP, but it suffices to understand that a non-trivial amount of storage is allocated for each half-open connection.

Systems typically limit the number of simultaneous half-open connections allowed. If this limit were not enforced, another denial of service attack would be possible by opening so many connections that all memory on the system is allocated.

It is tempting to label the SYN flood vulnerability as a part of the TCP specification, but this is not true. The specification of TCP does not limit the number of simultaneous half-open connections possible nor does it make any design demands that lead to a limit. This choice is left to the designer of a TCP implementation. Because it is possible to design a TCP implementation so that SYN floods have a negligible effect, the error is in the design of the TCP implementation and not the protocol. An example of a design that avoids the SYN flood vulnerability is Linux's SYN-cookie approach. [T$^+$97] In Linux, when the number of available possible half-open connections runs low, the amount of information stored about a connection request is reduced by hashing the connection's identifying information and storing the hash in a table. Because the hash size is small, many more hashes can be stored than TCP control blocks. The hash is then used as the sequence number for the reply packet. While SYN cookies do help to make a system less vulnerable to attack, they violate the TCP standard because TCP options on the SYN get ignored when an attack is in progress.

A brute-force design approach is to simply increase the number of possible half open connections. Hewlett Packard recommends this approach for its HP-UX operating system. [HPs97] This approach uses a great deal of memory but shows that an implementation can be designed so that it resists the attack.

2.5.5 Audit Requirements

The above analysis suggests a number of audit features. A host could audit information about the transitions taken in the implementation of the TCP state machine relating to half-open connections. This would include auditing every transition from the LISTEN state to the SYN_RECVD state. Also, every time that a connection establishment timer expires causing a transition from the SYN_RECVD state to the LISTEN state. Finally, transitions from the SYN_RECVD state to the ESTABLISHED state should be audited. From this information, the current number of half-open connections can be computed by scanning the log, and a SYN flood attack can be detected by comparing the number of half-open connections to maximum number of simultaneous half-open connections. Additionally, we could audit all state transitions in the TCP finite state machine. This may allow us to detect other attacks that could be based upon vulnerabilities in the TCP finite state machine.

An alternative collection of audit requirements would simply include the allocation and deallocation of resources for half-open connections. Also, for stateless detection of an attack, the trail might include the current number of new half-open connections that are possible. Allocation information would be useful for statelessly detecting a SYN flood. It may be that all of an implementation's resource allocation should be audited to help detect other resource starvation attacks.

2.6 The Land Vulnerability in TCP

In the following section, I will present a denial of service vulnerability in the TCP layer called "Land". Land was described in [Cen97]. The vulnerability, which affects both UNIX and non-UNIX hosts, causes some operating systems to stop responding to input, and causes others to slow significantly. I will describe the details of a Land attack and how it affects vulnerable hosts. I will also specify a simple attack signature that can be used to detect Land attacks. Finally, I will present an analysis of the vulnerability and show that the vulnerability is not an error in the implementation, but rather an error in the specification of the protocol.

2.6.1 The Land Attack

As with the SYN flood vulnerability, a Land attack targets the the TCP three-way handshake. (For a review of the three-way handshake, see section 2.5) The attack consists of sending a specially formed TCP connection request to a target computer T. The connection request packet has its SYN bit set, and the source and destination IP addresses are both assigned T's IP address. Additionally, the source and destination TCP ports are assigned the value of some port that is being listened to on T.

In short, the packet appears to be a connection request from T to T on the same TCP port. The result on some machines, such as those running Free-BSD or IRIX, is that the operating system no longer responds to input and hangs. On other operating systems such as Novel 4.11, the attack causes the system to slow significantly. [One97]

2.6.2 Analysis of the Vulnerability

To see why the vulnerability exists, we will look at [Pos81], the RFC defining TCP. The problem is that [Pos81] defines two possible courses of action when generating the response that T generates to the SYN packet described above. The first possibility is discussed on page 69 of [Pos81] and is the part of the RFC implemented by operating systems vulnerable to Land. The second possibility, discussed on page 36 of the RFC, would send a RST packet and would prevent the vulnerability. This ambiguity in the specification explains why over 30 different TCP/IP stacks contain the vulnerability. [One97]

In the first scenario (shown in figure 5), T receives the spoofed SYN packet with sequence number x and makes the transition to the SYN_RECVD state. T's response to the initial SYN packet is a SYN-ACK packet bound back for T with sequence number y and and an acknowledgement number of x+1. T is presented with this SYN-ACK packet, and page 69 of [Pos81] states that when an ACK packet outside the sequence number window is received while in the SYN_RECVD state that the ACK should be acknowledged with the acknowledgement number that is expected next. Following this part of the specification, vulnerable systems send an ACK with number x+1. The stack is now caught in a loop because when the stack receives the new ACK, it will send another ACK with the same content. Because many systems have interrupts disabled during this processing, their kernels are caught in an infinite loop. Systems that do not disable interrupts experience a degradation of service caused by the CPU cycles wasted in the infinite loop, but they do not hang.

The second portion of the RFC that applies to a Land attack deals with when TCP should send a RST packet. According to page 36 of [Pos81],

> If the connection is in any non-synchronized state (LISTEN, SYN-SENT, SYN-RECEIVED), and the incoming segment acknowledges something not yet sent (the segment carries an unacceptable ACK), ... a reset is sent.

If this portion of the RFC is implemented, the a RST packet would be sent in response to the SYN-ACK packet because the ACK number is unacceptable. The RST causes the TCP state to change back to LISTEN, and no further packets are generated. So, this approach would avoid the infinite loop and the vulnerability. See Figure 6.

2.6.3 Signature for the Land Attack

The signature for Land is based upon assembled IP datagrams because overlapping datagrams may change the value of the TCP SYN bit. The signature is:

> Does an assembled datagram have the following properties: the datagram's protocol field must specify TCP and must contain a valid TCP packet, the source address must equal the destination address, the TCP source port must equal the TCP destination port, and the TCP SYN bit must be set.

2.6.4 Audit Requirements for the Land Attack

The signature is based upon valid assembled IP datagrams that contain valid TCP packets. The required audit data are the source address, destination address, TCP code bits, TCP source port, and the TCP destination port. A valid IP datagram is well-formed and will be processed further by the system. Similarly, a valid TCP packet is one that is well formed and will be accepted by the system.

2.7 A Broadcast ICMP Attack and Variants

In this section we will discuss an attack based upon Internet Control Message Protocol (ICMP) messages that allows an attacker to consume excessive amounts of a victim network's bandwidth. [Cen98] The attack, popularly called "Smurf," uses a third party network to send large numbers of ICMP echo replies to a victim's network thereby exhausting the victim's network bandwidth or overloading a target host on the network. First, we discuss the features of ICMP and IP that facilitate the attack. Next, we describe the attack and give signatures that can be used to detect a Smurf attack. We also present arguments about the nature of the Smurf vulnerability.

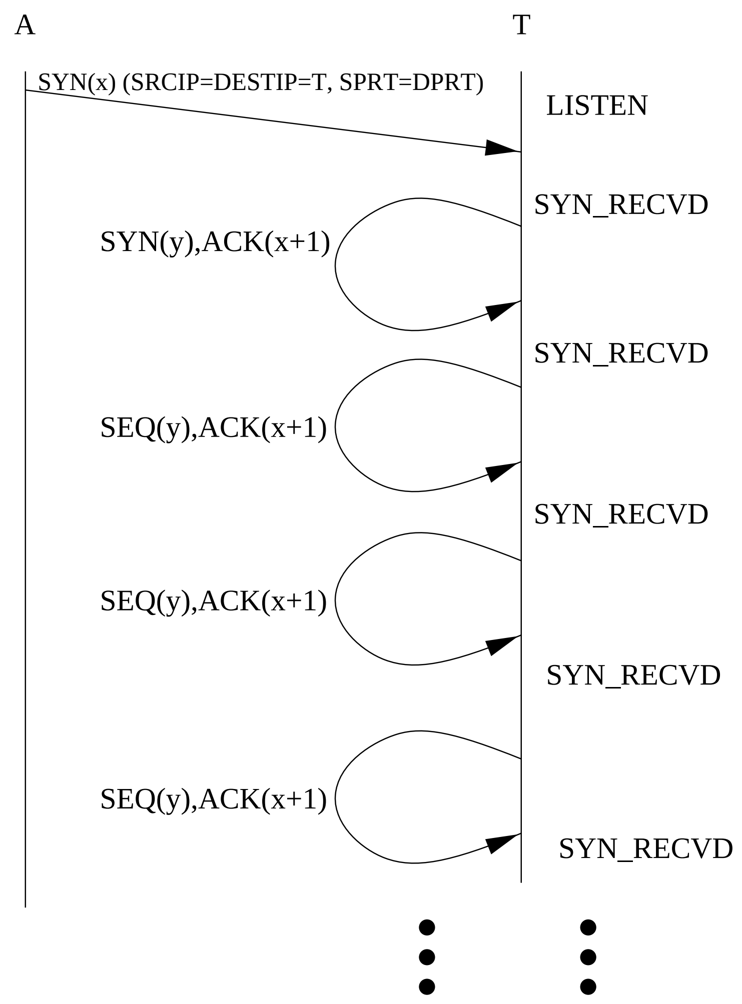

Figure 5. A specially crafted SYN packet causes a TCP three-way handshake to go into an infinite loop.

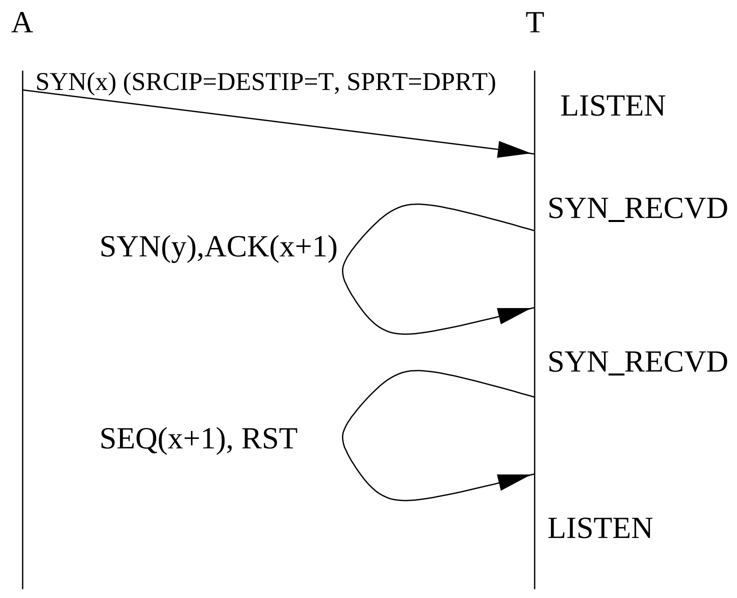

Figure 6. Page 36 of the RFC states that a RST packet should be sent in response to an unexpected ACK. If the implementation is such that a RST is sent, the vulnerability does not exist.

2.7.1 A Description of ICMP Echo Messages

ICMP is a protocol used by network hosts and routers to send error and control messages to each other. One type of ICMP message is the ICMP echo request message that allows programs such as `ping` to send an "are you there?" message to another host. The targeted host should reply to the source address of the echo request with another ICMP message, namely an ICMP echo reply. When the original host receives the echo reply, it knows that the target host is connected to the network and is responding to IP traffic. Even though an ICMP message is encapsulated in an IP packet, ICMP is not a higher level protocol than IP. It is a required part of an IP implementation and is encapsulated in IP packets so that ICMP messages can be sent over heterogeneous networks.

To illustrate the use of ICMP echo messages more clearly, let us say that machine A needs to confirm that another host, B, is connected to an IP network. A can send an ICMP echo request to B by forming an IP packet with B's IP address as its destination and A's address in the source field. Inside the IP packet's data area is an ICMP echo request message, and the packet is differentiated from non-ICMP traffic by setting the protocol field in the IP header to a value representing ICMP. When B receives an ICMP echo echo request, it creates an ICMP echo reply packet with itself as source and the destination address set to A's address. The echo reply packet is a copy of the ICMP echo request with its type field changed to indicate that it is a reply. B then sends the reply.

The attack uses two other features of IP during the attack: IP address forgery and IP broadcast addresses. IP address forgery is often used by an attacker to make tracing the source of an IP packet difficult. Typically, an attacker can simply create an IP datagram with an incorrect source address. Because some routers do not or can not ensure the validity of a source address, a forged packet is usually delivered to its destination. (Validity of a source address implies that a packet with a given source address could arrive at a given router input port.) [Mog84] specifies an IP addressing convention and algorithm that provides for broadcasting IP datagrams to an IP subnet. An example of such an address is 192.168.10.255 that would designate that a packet be sent to all hosts on the 192.168.10 subnet. This allows a host to send an IP datagram to all hosts on another subnet using only one datagram.

2.7.2 A Description of the Broadcast ICMP Attack

To describe the attack, we will refer to an attacker called A, a target denoted T, and a third party subnet of computers called N. A's goal is to send so much network traffic to T that T's network bandwidth is completely used up. The most naive approach to achieving this goal is for A to use a high bandwidth connection to the Internet to send large amounts of data to T. This approach has two problems: A needs an expensive, high bandwidth network connection, and T will realize that A is attacking T's network and use firewall mechanisms to block incoming data from A.

To provide anonymity and thwart T's attempts to filter out A's attack, A can send packets with forged IP source addresses to T. A can even randomly change the source address of each packet to make it impossible for T to filter by packet address.

A still has the other problem—he must have a network connection that is at least as fast as T's to send enough data to exhaust T's Internet connection. A uses the intermediate network, N, and IP broadcasts to overcome this problem. A constructs an ICMP echo request packet whose source address is T's address. A sets the source address to the N's broadcast address. A then sends this packet x times. One round of the attack is shown in figure 7.

The result of one packet depends on how many hosts are accessible on N and how many of them will reply to a broadcast ICMP message. If n hosts are reachable on N, each of them will send an ICMP echo reply to T. The result is that T receives nx ICMP echo replies that all come from machines on N. Hosts on N see ICMP echo requests from T. Neither hosts on N nor hosts on T can discern that A is the attacker. Also, because N must send as many replies as T receives, N's bandwidth is being consumed as well. Of course, A could use multiple intermediary subnets instead of only N so that the intermediaries might not even notice the attack.

By using intermediaries, A must only have 1/n as much bandwidth as T to perform the attack. Because a class C subnet can accommodate at least 250 hosts, n can be quite significant. Of course, we assume the N subnet's connection to the Internet is sufficiently fast to deliver the replies.

T may not be the only party whose service is disrupted by a Smurf attack. Because all of the ICMP echo replies generated by N go to T, a Smurf attack can consume all of N's Internet connection bandwidth. Because of this, it is valuable to also consider the signature and audit concerns of the intermediary.

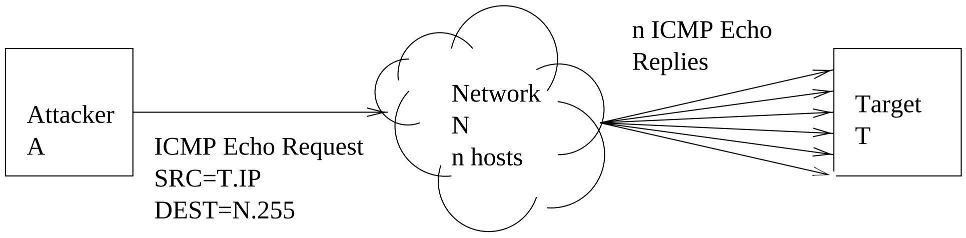

Figure 7. A single ICMP echo request broadcast to N generates n replies to T. The echo request's source address has been changed to T.

2.7.3 A Signature for Smurf Attacks

We first consider signatures useful for detecting that a network is the target of the attack. A simple signature for a Smurf attack would count the number of unsolicited ICMP echo replies a host receives in a fixed time period and compare it to some threshold value. This signature is:

Number of echo replies accepted during a time period > threshold

One problem with this approach is that ICMP echo messages can be of varying lengths. Large ICMP packets are fragmented when traversing a network and could use considerably more network bandwidth than the typical, unfragmented ICMP packet. In this case, network bandwidth may be exhausted, but because we would only be counting assembled ICMP packets, the threshold may not be exceeded.

A more robust signature would total the lengths of all unsolicited ICMP echo replies over some time interval.

Number of bytes of ICMP echo reply traffic received during a time period > threshold

The total number of bytes could then be compared to a predetermined threshold. Because we are measuring the number of bytes used for unsolicited ICMP echo reply traffic over a time interval, fragments are taken into account. The previous two signatures were based upon "unsolicited" ICMP echo replies. We assume for now that the audit trail has sufficient information to determine what echo requests have been sent. We will cover this in the audit requirements section below.

We now present a signature for detecting that a host is being used as a Smurf intermediary. When they are being used to "Smurf" T, hosts on N will receive ICMP echo requests that have an IP broadcast address for their destination address. Additionally, the datagrams will have T's address as their source address. One possible signature is

If a valid ICMP echo request is received with a destination address that is an IP broadcast address, then true.

2.7.4 An Analysis of the Vulnerability

The vulnerability that is exploited by the Smurf attack differs only slightly from that exploited by any other flood vulnerability. If you disregard the intermediate network, you can see that an attacker with a fast enough connection to the Internet and a network of machines he controls can mount the same attack by having each machine send unsolicited ICMP echo replies to the target. If this is the case, then the packets sent to the target could be arbitrary ones as there is no need to use the multiplying power of an intermediate network.

From the above analysis, we conclude that neither IP spoofing nor the use of ICMP is fundamental to the vulnerability. However, IP spoofing is still needed to maintain anonymity. The real vulnerability seems to relate to the user's expectations of the network and how they are not upheld by the facilities provided by the network. While IP provides best effort packet delivery, it makes no guarantees about availability of network resources.

2.7.5 Audit Requirements

The audit requirements needed to detect Smurf attacks are based upon the signatures given earlier. To support evaluation of the target signature, an audit trail must note that a valid ICMP echo reply was received and include its length. Furthermore, because a rate is being computed, a timestamp must be included. Also, a system administrator or IDS may need to know the source address of the echo reply so that the intermediary network personnel can be notified. Additionally, an IDS might temporarily block ICMP traffic from the intermediary to reduce the effects of the attack on the target host.

To support evaluation of the target signature, we need to be able to differentiate solicited ICMP echo replies from those that are unsolicited. This could be done by including ICMP echo requests sent by the host along with a timestamp in the audit trail. A solicited echo reply is one that matches an earlier echo request within some timeout period. The ping program could be modified to log this information.

For evaluating the intermediary signature, we must audit valid ICMP echo requests received by the host. Again because we may need to measure a rate, a timestamp is required. Also required are the source and destination addresses and the ICMP type fields.

3 General Host-Based Audit Data Requirements

This section summarizes and expands upon the audit data requirements proposed in the preceding chapter. From these requirements some suggestions can be made about what should be audited to detect yet unknown attacks.

3.1 Summary of Audit Requirements from Our Analysis

All data listed here is assumed to be from valid frames and packets. Valid packets are those that arrive at the host and would be accepted for processing. An IP datagram with an incorrect checksum value is one example of an invalid packet. Similarly, a TCP packet with a bad checksum would also be considered invalid. Another example would be an IP datagram that could not be assembled because a fragment was missing, but this does not imply that the fragments are invalid–each of them could be valid in their own right. The following are audit requirements specified in section 2.

- From valid ARP reply frames
  - Sender's Hardware Address
  - Sender's IP address
  - Network Interface from which the ARP frame originated
- From valid pre-assembly IP datagrams
  - source address
  - destination address
  - protocol field
  - fragment offset
  - length
  - header length
  - MF bit
  - identification
- From IP fragment reassembly module
  - Notification of fragment overlap
  - Content of fragments involved in the overlap plus the other fragment information associated with the datagram being assembled.
- From valid ICMP messages
  - IP Source address
  - IP Target Address
  - ICMP Type field
  - ICMP Identifier
  - ICMP Sequence Number

- From valid TCP packets
  - IP Source Address
  - IP Destination Address
  - TCP Source Port
  - TCP Destination Port
  - TCP Code Bits
- From the TCP implementation
  - Transitions from LISTEN to SYN_RECVD state
  - Transitions from SYN_RECVD to LISTEN state
  - Transitions from SYN_RECVD to CONNECTED state
  - Upon allocation of a half-open connection, the number of possible half-open connections still possible.

3.2 General Audit Requirements for IP Implementations

The audit items listed above are specific to the attacks that we have analyzed in this paper. However, they do suggest guidelines that may be applied to the selection of audit content for low-level network systems. Here we will try to make some generalizations about what should be audited from a low-level protocol stack.

3.2.1 Headers

The first general audit content we suggest is that header data be audited at several points throughout an IP implementation. In the list above, we see that most header fields of ARP, IP (pre-assembly), TCP, and ICMP packets are useful. Also, note that data payloads are only in the list under exceptional circumstances. From these observations we suggest that all protocol header data be audited.

Additionally, the data suggest that valid packet data payloads can be left out of the trail unless the data would be overwritten or thrown away by the protocol implementation. Such is the case in the IP Fragmentation vulnerability discussed in section 2.2. The data that would be overwritten by the overlapping fragments should be audited so that further analysis can be done to determine if the overlap might have been malicious or the result of anomalous behavior of a network component. Similarly, Teardrop attacks from section 2.3 cause correct IP implementations to overwrite pre-existing network data.

Another argument for auditing only header information is that protocol stacks are designed to transfer arbitrary data to and from the application layer. Typically, the only interaction that low-level protocols have with the data is to verify checksums. Our notion of a valid packet makes the data payloads unnecessary for checksum verification. Also, because data payloads are passed up to the application layer, conventional system call audit data should include data payloads when they are read from the operating system.

3.2.2 Resource Allocation

Section 2.5, describing the SYN Flood attack, makes a case for auditing the allocation and deallocation of fixed resources that an external agent can cause the protocol implementation to consume. In the case of SYN Flood, an attacker causes the TCP implementation to use all of the resources set aside for managing half-open connections. When these resources are exhausted, the host cannot accept any more TCP connections. Other code in a protocol stack that allocates resources from a fixed pool should be audited. Another example might be an implementation that uses a fixed number of buffers for storing IP fragments while they wait for assembly. The allocation of these buffers should be audited because an attacker might be able to keep these buffers full by sending fragmented traffic and thereby keep the target host from reassembling non-malicious fragmented traffic.

3.2.3 TCP State Machine Transitions

TCP is a complex protocol that is modeled and specified according to a finite state machine. As seen with the Land (Section 2.6) and SYN Flood (Section 2.5) attacks, neither the specification nor the implementation of TCP is fully understood with regards to security. Because of this, it may be useful to log transitions made in the TCP state machine along with the event that triggers the transition. Because TCP is by far the most complex of the low-level IP protocols, it is likely that more vulnerabilities exist in both the TCP state machine and its various implementations. Having a record of the transitions made along with the current state of the machine may allow us to detect future vulnerabilities that may arise.

3.2.4 Protocols

The protocols for which audit data is listed are TCP, ICMP, IP, and ARP. Notably missing are UDP and IGMP, but this does not imply that these should not be audited. We suggest that headers be audited for these as well as any other protocols that can properly be considered at the network or transport layer.

3.2.5 Breaking Layering

Our primary suggestion has been to audit the headers of packets, but an audit record that contains the header of only that layer may be fairly useless. An example of this is the TCP Land vulnerability described in Section 2.6. If we audit the TCP header separately from the IP header, we are left without the source and destination IP addresses needed to evaluate the signature. Similarly, it may be useful to know details about the headers of link layer frames in which packets arrive. This could be useful for determining if an attack is originating on the LAN or remotely.

Because headers need to be related across layers, some mechanism in the audit trail must link related headers so signatures that rely upon headers at different levels may be evaluated. This may be accomplished at the expense of increasing the size of the audit trail by including lower-level headers in the audit record. For example, when a TCP packet is being processed, an audit record would be generated with the TCP header information, but it would also include the IP header and perhaps even an Ethernet header. Instead of including the lower-level headers, the record could refer to earlier records in the trail. This would tend to make the IDS maintain more state during detection. Further work must be done to determine how to best represent header data in a trail as it becomes even more complicated when considering the many-to-one nature of fragmentation.

4 Conclusion

In this paper, we have defined low-level network vulnerabilities and differentiated them from network-based vulnerabilities. We have analyzed and proposed host-based detection methods for seven known low-level vulnerabilities. From our analysis of the vulnerabilities and their attack signatures, we have proposed new audit events that could be added to operating system audit trails.

New audit events should be added for all low-level protocols in a protocol stack, but data payloads should not be audited at these levels except in cases where valid data is being overwritten or thrown away. This implies that protocol header information should comprise the bulk of the new audit items.

Another area where audit events could be added is resource allocation and availability. When the system can be caused to allocate resources by an attacker, an audit record should be generated. Similarly, deallocation of resources should also be audited. To facilitate stateless detection of resource starvation attacks, audit records should be generated when resources are exhausted.

Finally, in any complex piece of software, such as that which implements the TCP state machine, there may be errors that are very difficult to detect without some amount of internal auditing. One approach is then to audit behavior of the software so as to help verify that it is behaving according to specifications. Because of this, TCP state transitions should be audited.

By adding these new audit items to an IP implementation, designers of host-based intrusion detection systems can more easily detect and react to low-level network attacks. By detecting these attacks at individual hosts, we decrease the chances of network-wide failure of the intrusion detection system.

References

[Arc97] Rootshell Archive. Linux IP Fragmentation Bug - Teardrop. `http://www.rootshell.com/archive-ld8dkslxlxja/199711/teardrop.linux`, November 1997.

[Atk95] R. Atkinson. *RFC 1826 IP Encapsulating Security Payload (ESP).* Network Working Group, 1995.

[Cen96a] CERT Coordination Center. CERT advisory CA-96.26:denial-of-service attack via ping. `ftp://ftp.cert.org/pub/cert_advisories/CA-96.26.ping`, December 1996.

[Cen96b] CERT Coordination Center. TCP SYN flooding and IP spoofing attacks, September 1996. CA-96:21.

[Cen97] CERT Coordination Center. CERT advisory CS-97.28 - teardrop_land. `ftp://ftp.cert.org/pub/cert_advisories/CA-97.28.Teardrop_Land`, December 1997.

[Cen98] CERT Coordination Center. "smurf" ip denial-of-service attacks, January 1998. CA-98.01.

[Com95] D. E. Comer. *Internetworking with TCP/IP*. Prentice-Hall, third edition, 1995.

[Cra95] Cray Research, Inc. *UNICOS Multilevel Security (MLS) Feature User's Guide*, August 1995. SG-2111 9.0.

[Dig96] Digital Equipment Corporation, Maynard Massachusetts. *OpenVMS Guide to System Security*, November 1996. OpenVMS VAX Version 7.1.

[HDL+90] L. Todd Heberlein, Gihan V. Dias, Karl N. Levitt, Biswanath Mukherjee, Jeff Wood, and David Wolber. A network security monitor. In *Proceedings of the 1990 IEEE Symposium on Research in Security and Privacy*, Oakland, CA, May 1990.

[Hew95] Hewlett Packard Company, Palo Alto, California. *HP-UX System Administration Tasks*, 2 edition, June 1995. B2355-90079.

[HJS+93] Judith Hochberg, Kathleen Jackson, Cathy Stallings, J. F. McClary, David DuBois, and Josephine Ford. NADIR: An automated system for detecting network intrusion and misuse. *Computers and Security*, 12(3):274–283, May 1993.

[HPs97] Hewlett-packard security bulletin: HPSBUX9704-060, April 1997.

[Ilg92] Koral Ilgun. USTAT, a real-time intrusion detection system for UNIX. Master's thesis, University of California, Santa Barbara, November 1992.

[JP81] Ed. Jon Postel. *RFC-791 Internet Protocol DARPA Internet Program Protocol Specification.* Information Sciences Institute, September 1981.

[Ken] Malachi Kenney. It's the ping o' death page! `http://www.sophist.demon.co.uk/ping/`.

[Krs98] Ivan Victor Krsul. *Computer Vulnerability Analysis*. PhD thesis, Purdue University, 1998.

[KST98] Ivan Krsul, Eugene Spafford, and Mahesh Tripunitara. Computer vulnerability analysis. Technical report, COAST Laboratory, 1998.

[Kum95] Sandeep Kumar. *Classification and Detection of Computer Intrusions*. PhD thesis, Purdue University, August 1995.

[Mic95] Microsoft Corporation. *Microsoft Win32 Programmer's Reference*, 1995.

[Mog84] Jeffrey Mogul. *RFC-922 Broadcasting Internet Datagrams in the Presence of Subnets.* Network Working Group, October 1984.

[One97] Aleph One. Re: LAND attack update. BUGTRAQ Listserv Archived at `http://www.geek-girl.com/bugtraq/1997_4/0380.html`, November 1997.

[Plu82] David C. Plummer. *RFC-826 An Ethernet Address Resolution Protocol.* Network Working Group, November 1982 1982.

[PN98] Thomas H. Ptacek and Timothy N. Newsham. Insertion, evasion, and denial of service: Eluding network intrusion detection. Technical report, Secure Networks Inc, January 1998.

[Pos81] J. Postel, editor. *RFC-793 Transmission Datagram Protocol*. Information Sciences Institute, USC, CA, September 1981.

[Pri97] Katherine E. Price. Host-based misuse detection and conventional operating systems' audit data collection. Master's thesis, Purdue University, December 1997.

[SKK$^+$97] Christoph L. Schuba, Ivan Krsul, Markus Kuhn, E. H. Spafford, Aurobindo Sundaram, and Diego Zamboni. Analysis of a denial of service attack on TCP. In *IEEE Symposium on Security and Privacy*. IEEE, May 1997.

[Sun95] Sun Microsystems, Inc., Mountain View, California. *SunSHIELD Basic Security Module Guide*, November 1995.

[Sys] Internet Security Systems. *Real-time attack recognition and response: A solution for tightening network security*. Atlanta, GA. Available at: `http://www.iss.net/prod/whitepaper.html`.

[T$^+$97] Linus Torvalds et al. Linux 2.0.31. Available at `http://www.kernel.org/pub/linux/kernel/v2.0/linux-2.0.31.tar.gz`, October 1997.

[Tan88] Andrew S. Tannenbaum. *Computer Networks.* Prentice Hall, Englewood Cliffs, NJ, 1988.

[Vol97] Yuri Volobuev. Playing redir games with ARP and ICMP. `http://www.rootshell.com/archive-ld8dkslxlxja/199711/arp_fun.txt/`, 1997.

[ZBGF$^+$98] Diego Zamboni, Jai Sundar Balasubramaniyan, Jose Omar Garcia-Fernandez, David Isacoff, and Eugene Spafford. Design of an architecture for intrusion detection using autonomous agents. Technical Report CTR 98/05, COAST Laboratory, Purdue University, West Lafayette, IN, May 1998.

[ZRT95] G. Ziemba, D. Reed, and P. Traina. *RFC-1858 Security Considerations for IP Fragment Filtering.* Network Working Group, October 1995.