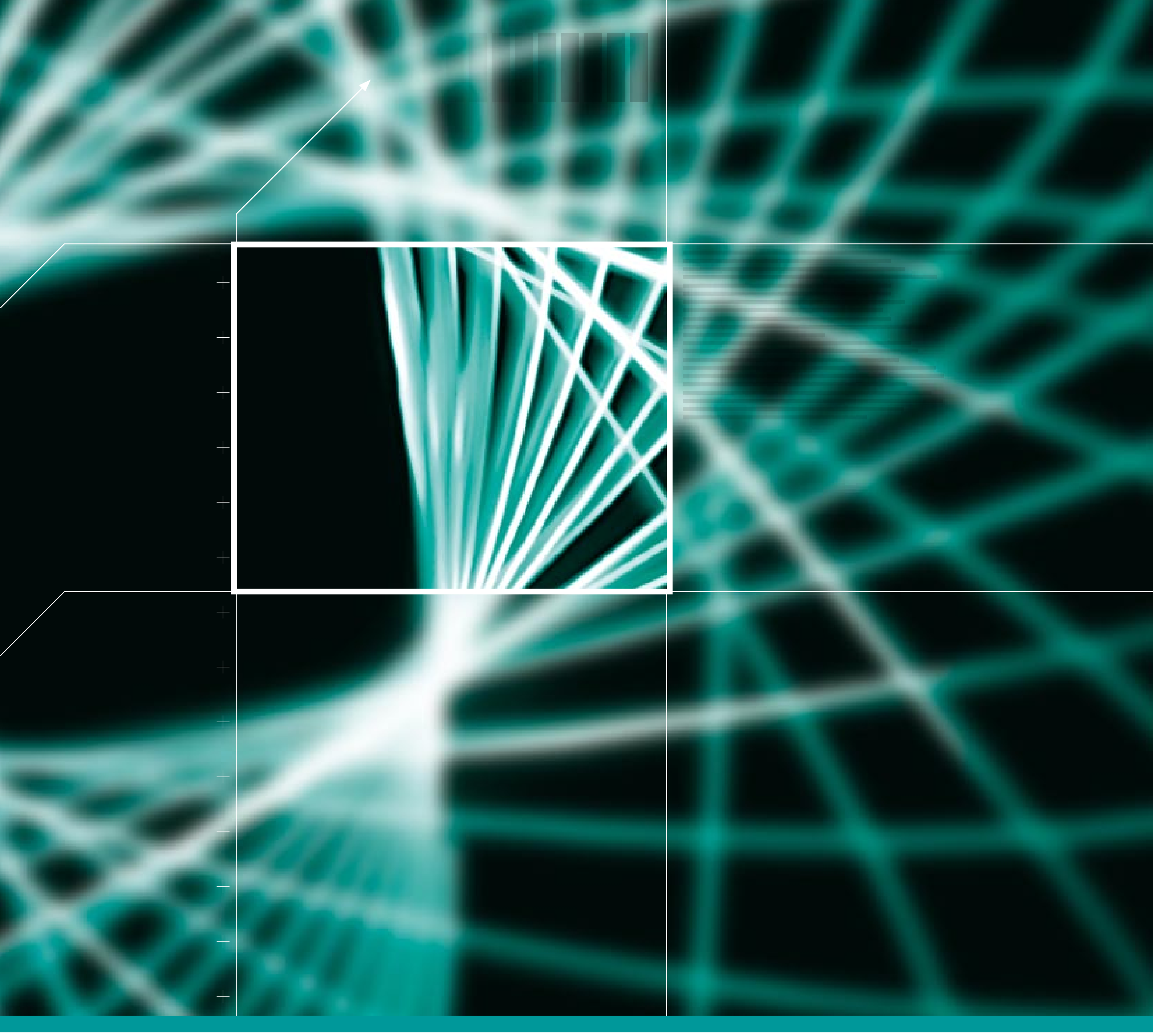

SECURITY ASSESSMENT OF THE INTERNET PROTOCOL

JULY 2008

CPNI
Centre for the Protection of National Infrastructure

Written by Fernando Gont on behalf of CPNI.

www.cpni.gov.uk

Table of Contents

1. Preface

1.1 Introduction

The TCP/IP protocols were conceived during a time that was quite different from the hostile environment they operate in now. Yet a direct result of their effectiveness and widespread early adoption is that much of today's global economy remains dependent upon them.

While many textbooks and articles have created the myth that the Internet Protocols (IP) were designed for warfare environments, the top level goal for the DARPA Internet Program was the sharing of large service machines on the ARPANET [Clark, 1988]. As a result, many protocol specifications focus only on the operational aspects of the protocols they specify and overlook their security implications.

Though Internet technology has evolved, the building blocks are basically the same core protocols adopted by the ARPANET more than two decades ago. During the last twenty years many vulnerabilities have been identified in the TCP/IP stacks of a number of systems. Some were flaws in protocol implementations which affect only a reduced number of systems. Others were flaws in the protocols themselves affecting virtually every existing implementation [Bellovin, 1989]. Even in the last couple of years researchers were still working on security problems in the core protocols [Gont, 2008] [Watson, 2004] [NISCC, 2004] [NISCC, 2005].

The discovery of vulnerabilities in the TCP/IP protocols led to reports being published by a number of CSIRTs (Computer Security Incident Response Teams) and vendors, which helped to raise awareness about the threats as well as the best mitigations known at the time the reports were published.

Much of the effort of the security community on the Internet protocols did not result in official documents (RFCs) being issued by the IETF (Internet Engineering Task Force) leading to a situation in which "known" security problems have not always been addressed by all vendors. In many cases vendors have implemented quick "fixes" to protocol flaws without a careful analysis of their effectiveness and their impact on interoperability [Silbersack, 2005].

As a result, any system built in the future according to the official TCP/IP specifications might reincarnate security flaws that have already hit our communication systems in the past.

Producing a secure TCP/IP implementation nowadays is a very difficult task partly because of no single document that can serve as a security roadmap for the protocols.

There is clearly a need for a companion document to the IETF specifications that discusses the security aspects and implications of the protocols, identifies the possible threats, proposes possible counter-measures, and analyses their respective effectiveness.

This document is the result of an assessment of the IETF specifications of the Internet Protocol from a security point of view. Possible threats were identified and, where possible, counter-measures were proposed. Additionally, many implementation flaws that have led to security vulnerabilities have been referenced in the hope that future implementations will not incur the same problems. This document does not limit itself to performing a security assessment of the relevant IETF specification but also offers an assessment of common implementation strategies.

Whilst not aiming to be the final word on the security of the IP, this document aims to raise awareness about the many security threats based on the IP protocol that have been faced in the past, those that we are currently facing, and those we may still have to deal with in the future. It provides advice for the secure implementation of the IP, and also insights about the security aspects of the IP that may be of help to the Internet operations community.

Feedback from the community is more than encouraged to help this document be as accurate as possible and to keep it updated as new threats are discovered.

1.2 Scope of this document

While there are a number of protocols that affect the way in which IP systems operate, this document focuses only on the specifications of the IP. For example, routing and bootstrapping protocols are considered out of the scope of this project.

The following IETF RFCs were selected for assessment as part of this work:

- RFC 791, “Internet Protocol. DARPA Internet Program. Protocol Specification” (51 pages).
- RFC 815, “IP datagram reassembly algorithms” (9 pages).
- RFC 1122, “Requirements for Internet Hosts -- Communication Layers” (116 pages).
- RFC 1812, “Requirements for IP Version 4 Routers” (175 pages).
- RFC 2474, “Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers” (20 pages).
- RFC 2475, “An Architecture for Differentiated Services” (36 pages).
- RFC 3168, “The Addition of Explicit Congestion Notification (ECN) to IP” (63 pages).

1.3 Organisation of this document

This document is basically organised in two parts: “Internet Protocol header fields” and “Internet Protocol mechanisms”. The former contains an analysis of each of the fields of the Internet Protocol header, identifies their security implications, and discusses the possible counter-measures. The latter contains an analysis of the security implications of the mechanisms implemented by the Internet Protocol.

1.4 Typographical conventions

Throughout this document the header fields of the Internet Protocols are discussed in detail. In some cases, a given term may have a slightly different meaning depending on whether it is used to refer to a concept or to a specific field of the Internet Protocol header. To avoid any possible confusion arising from this ambiguity, when referring to a specific field of the IP header the name of the field is written in this **font type**.

Throughout the document there are also a number of parenthetical notes such as this one, to provide additional details or clarifications.

1.5 Getting the latest version of this document

It is expected that revisions of this document will be published in response to the feedback provided by the community. Revisions of this document will be available at: www.cpni.gov.uk/Products/technical.aspx

1.6 Advice and guidance to vendors

Vendors are urged to contact CPNI's Vulteam (vulteam@cpni.gov.uk) if they think they may be affected by the issues described in this document. As the lead coordination center for these issues, CPNI is well placed to give advice and guidance as required.

CPNI works extensively with government departments and agencies, commercial organisations and the academic community to research vulnerabilities and potential threats to IT systems, especially where they may have an impact on Critical National Infrastructure's (CNI).

Other ways to contact CPNI, plus CPNI's PGP public key, are available at www.cpni.gov.uk.

1.7 Acknowledgements

This assessment was written by Fernando Gont on behalf of CPNI.

The author would like to thank Randall Atkinson, John Day, Juan Fraschini, Roque Gagliano, Guillermo Gont, Martin Marino, Pekka Savola, and Christos Zoulas for providing valuable comments on earlier versions of this document.

The author would like to thank Randall Atkinson and Roque Gagliano, who generously answered a number of questions.

Finally, the author would like to thank CPNI for their continued support.

2. The Internet Protocol

The IP provides a basic data transfer function, in the form of data blocks called "datagrams", from a source host to a destination host, across the possible intervening networks. It additionally provides some functions that are useful for the interconnection of heterogeneous networks, such as fragmentation and reassembly.

The "datagram" has a number of characteristics that makes it convenient for interconnecting systems [Clark, 1988]:

- It eliminates the need of connection state within the network, which improves the survivability characteristics of the network.
- It provides a basic service of data transport that can be used as a building block for other transport services (reliable data transport services, etc.).
- It represents the minimum network service assumption, which enables IP to be run over virtually any network technology.

3. Internet Protocol header fields

The IETF specifications of the Internet Protocol define the syntax of the protocol header, along with the semantics of each of its fields. Figure 1 shows the format of an IP datagram.

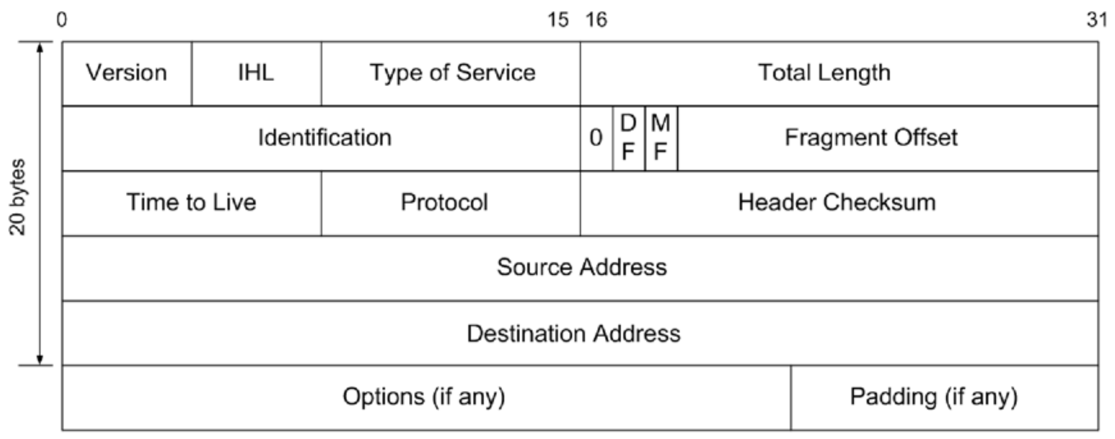

Figure 1: Internet Protocol header format

Even when the minimum IP header size is 20 bytes, an IP module might be handed an (illegitimate) “datagram” of less than 20 bytes. Therefore, before doing any processing of the IP header fields, the following check should be performed by the IP module on the packets handed by the link-layer:

link-layer.PayloadSize >= 20

If the packet does not pass this check it should be dropped.

The following subsections contain further sanity checks that should be performed on IP packets.

3.1 Version

This is a 4-bit field that indicates the version of the IP, and thus the syntax of the packet. For IPv4, this field must be 4.

When a link-layer protocol de-multiplexes a packet to an internet module, it does so based on a “Protocol Type” field in the data-link packet header.

In theory, different versions of IP could coexist on a network by using the same “Protocol Type” at the link-layer, but a different value in the Version field of the IP header. Thus a single IP module could handle all versions of the Internet Protocol, differentiating them by means of this field.

However, in practice different versions of IP are identified by a different “Protocol Type” number in the link-layer protocol header. For example, IPv4 datagrams are encapsulated in Ethernet frames using a “Protocol Type” field of 0x0800, while IPv6 datagrams are encapsulated in Ethernet frames using a “Protocol Type” field of 0x86DD [IANA, 2006a].

Therefore if an IPv4 module receives a packet, the **Version** field must be checked to be 4. If this check fails the packet should be silently dropped.

3.2 IHL (Internet Header Length)

The IHL (Internet Header Length) indicates the length of the internet header in 32-bit words (4 bytes). As the minimum datagram size is 20 bytes, the minimum legal value for this field is 5 and the following check should be enforced:

IHL >= 5

For obvious reasons, the Internet header cannot be larger than the whole Internet datagram it is part of and therefore the following check should be enforced:

IHL * 4 <= **Total Length**

The above check allows for Internet datagrams with no data bytes in the payload that, while nonsensical for virtually every protocol that runs over IP, is still legal.

3.3 TOS

Figure 2 shows the syntax of the **Type of Service field**, defined by RFC 791 [Postel, 1981], and updated by RFC 1349 [Almquist, 1992].

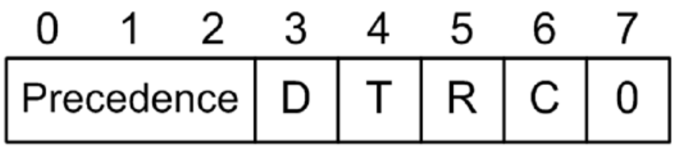

Figure 2: Type of Service field

Bits	0-2:	Precedence.
Bit	3:	0 = Normal Delay, 1 = Low Delay.
Bits	4:	0 = Normal Throughput, 1 = High Throughput.
Bit	5:	0 = Normal Reliability, 1 = High Reliability.
Bit	6:	0 = Normal Cost, 1 = Minimise Monetary Cost
Bits	7:	Reserved for Future Use (must be zero).

Where:
Precedence

111:	Network Control
110:	Internetwork
101:	CRITIC/ECP
100:	Flash Override
011:	Flash
010:	Immediate
001:	Priority
000:	Routine

The **Type of Service** field can be used to affect the way in which the packet is treated by the systems of a network that process it. Section 4.2.1 ("Precedence-ordered queue service") and Section 4.2.3 ("Weak TOS") of this document describe the security implications of the **Type of Service** field in the forwarding of packets.

3.4 Total Length

The **Total Length** field is the length of the datagram, measured in bytes, including both the IP header and the IP payload. Being a 16-bit field, it allows for datagrams of up to 65535 bytes. RFC 791 [Postel, 1981] states that all hosts should be prepared to receive datagrams of up to 576 bytes (whether they arrive as a whole or in fragments). However, most modern implementations can reassemble datagrams of at least 9 Kbytes.

Usually a host will not send to a remote peer an IP datagram larger than 576 bytes unless it is explicitly signaled that the remote peer is able to receive such "large" datagrams (for example, by means of TCP's MSS option). However, systems should assume that they may be sent datagrams larger than 576 bytes regardless of whether they signal their remote peers to do so or not. In fact it is common for NFS [Shepler et al, 2003] implementations to send datagrams larger than 576 bytes, even without explicit signaling that the destination system can receive such "large" datagram.

Additionally, see the discussion in Section 4.1 "Fragment reassembly" regarding the possible packet sizes resulting from fragment reassembly.

Implementations should be aware that the IP module could be handed a packet larger than the value actually contained in the Total Length field. Such a difference usually has to do with legitimate padding bytes at the link-layer protocol, but it could also be the result of malicious activity by an attacker. Furthermore, even when the maximum length of an IP datagram is 65535 bytes, if the link-layer technology in use allows for payloads larger than 65535 bytes an attacker could forge such a large link-layer packet, meaning it for the IP module. If the IP module of the receiving system were not prepared to handle such an oversized link-layer payload an unexpected failure might occur. Therefore, the memory buffer used by the IP module to store the link-layer payload should be allocated according to the payload size reported by the link-layer, rather than according to the Total Length field of the IP packet it contains.

The IP module could also be handled a packet that is smaller than the actual IP packet size claimed by the **Total Length** field. This could be used, for example, to produce an information leakage. Therefore the following check should be performed:

Link-layer.PayloadSize >= **Total Length**

If this check fails the IP packet should be dropped. As the previous expression implies, the number of bytes passed by the link-layer to the IP module should contain at least as many bytes as claimed by the **Total Length** field of the IP header.

[US-CERT, 2002] is an example of the exploitation of a forged IP **Total Length** field to produce an information leakage attack.

3.5 Identification (ID)

The **Identification** field is set by the sending host to aid in the reassembly of fragmented datagrams. At any time, it needs to be unique for each set of {**Source Address, Destination Address, Protocol**}.

In many systems the value used for this field is determined at the IP layer on a protocol-independent basis. Many of those systems also simply increment the IP **Identification** field for each packet they send.

This implementation strategy is inappropriate for a number of reasons. First, if the **Identification** field is set on a protocol-independent basis, it will wrap more often than necessary, and thus the implementation will be more prone to the problems discussed in [Kent and Mogul, 1987] and [Heffner, Mathis, and Chandler, 2006].

Secondly, this implementation strategy opens the door to an information leakage that can be exploited to in a number of ways. [Sanfilippo, 1998a] originally pointed out how this field could be examined to determine the packet rate at which a given system is transmitting information. Later, [Sanfilippo, 1998b] described how a system with such an implementation can be used to perform a stealth port scan to a third (victim) host. [Sanfilippo, 1999] explained how to exploit this implementation strategy to uncover the rules of a number of firewalls. [Bellovin, 2002] explains how the IP **Identification** field can be exploited to count the number of systems behind a NAT. [Fyodor, 2004] is an entire paper on most (if not all) the ways to exploit the information provided by the **Identification** field of the IP header.

3.5.1 Some workarounds implemented by the industry

As the IP **Identification** field is only used for the reassembly of datagrams, some operating systems (such as Linux) decided to set this field to 0 in all packets that have the DF bit set. This would, in principle, avoid any type of information leakage. However, it was detected that some non-RFC-compliant middle-boxes fragmented packets even if they had the DF bit set. In such a scenario all datagrams originally sent with the DF bit set would result in fragments that would have an **Identification** field of 0, which would lead to problems (“collision” of the Identification number) in the reassembly process.

Linux (and Solaris) later set the IP **Identification** field on a per-IP-address basis. This avoids some of the security implications of the IP **Identification** field, but not all. For example, systems behind a load balancer can still be counted.

3.5.2 Possible security improvements

Contrary to common wisdom, the IP **Identification** field does not need to be system-wide unique for each packet, but has to be unique for each {**Source Address, Destination Address, Protocol**} tuple.

For instance, the TCP specification defines a generic send() function which takes the IP ID as one of its arguments.

We provide an analysis of the possible security improvements that could be implemented, based on whether the protocol using the services of IP is connection-oriented or connection-less.

Connection-oriented protocols

To avoid the security implications of the information leakage described above, a pseudo-random number generator (PRNG) could be used to set the IP **Identification** field on a {**Source Address, Destination Address**} basis (for each connection-oriented transport protocol).

[Klein, 2007] is a security advisory that describes a weakness in the pseudo random number generator (PRNG) in use for the generation of the IP **Identification** by a number of operating systems.

While in theory a pseudo-random number generator could lead to scenarios in which a given **Identification** number is used more than once in the same time-span for datagrams that end up getting fragmented (with the corresponding potential reassembly problems), in practice this is unlikely to cause trouble.

By default, most implementations of connection-oriented protocols, such as TCP, implement some mechanism for avoiding fragmentation (such as the Path-MTU Discovery mechanism [Mogul and Deering, 1990]). Thus, fragmentation will only take place sporadically when a non-RFC-compliant middle-box is placed somewhere along the path that the packets travel to get to the destination host. Once the sending system is signaled by the middle-box that it should reduce the size of the packets it sends, fragmentation would be avoided. Also, for reassembly problems to arise, the same **Identification** field should be frequently reused and either strong packet reordering or packet loss should take place.

Nevertheless, regardless of what policy is used for selecting the **Identification** field, with the current link speeds fragmentation is already bad enough to rely on it. A mechanism for avoiding fragmentation should be implemented instead.

Connectionless protocols

Connectionless protocols usually have these characteristics:

- lack of flow-control mechanisms,
- lack of packet sequencing mechanisms
- lack of reliability mechanisms (such as "timeout and retransmit").

This means that the scenarios and/or applications for which connection-less transport protocols are used assume that:

- Applications will be used in environments in which packet reordering is very unlikely (such as Local Area Networks), as the transport protocol itself does not provide data sequencing.
- The data transfer rates will be low enough that flow control will be unnecessary.
- Packet loss is not important and probably also unlikely.

With these assumptions in mind, the **Identification** field could still be set according to a pseudo-random number generator (PRNG). In the event a given **Identification** number was reused while the first instance of the same number is still on the network, the first IP datagram would be reassembled before the fragments of the second IP datagram get to their destination.

In the event this was not the case, the reassembly of fragments would result in a corrupt datagram. While some existing work [Silbersack, 2005] assumes that this error would be caught by some upper-layer error detection code, the error detection code in question (such as UDP's checksum) might be intended to detect single bit errors, rather than data corruption arising from the replacement of a complete data block (as is the case in corruption arising from collision of IP **Identification** numbers).

> In the case of UDP, unfortunately some systems have been known to not enable the UDP checksum by default. For most applications, packets containing errors should be dropped. Probably the only application that may benefit from disabling the checksum is streaming media, to avoid dropping a complete sample for a single-bit error.

In general, if IP **Identification** number collisions become an issue for the application using the connection-less protocol, then use of a different transport protocol (which hopefully avoids fragmentation) should be considered.

An attacker could intentionally exploit collisions of IP **Identification** numbers to perform a Denial of Service attack by sending forged fragments that would cause the reassembly process to result in a corrupt datagram, that would either be dropped by the transport protocol or would incorrectly be handed to the corresponding application. This issue is discussed in detail in section 4.1 ("Fragment Reassembly").

3.6 Flags

The IP header contains 3 control bits, two of which are currently used for the fragmentation and reassembly function.

As described by RFC 791, their meaning is:

Bit 0:	reserved, must be zero
Bit 1:	(**DF**) 0 = May Fragment, 1 = Don't Fragment
Bit 2:	(**MF**) 0 = Last Fragment, 1 = More Fragments

The **DF** bit is usually set to implement the Path-MTU Discovery (PMTUD) mechanism described in [Mogul and Deering, 1990]. However, it can also be exploited by an attacker to evade Network Intrusion Detection Systems. An attacker could send a packet with the **DF** bit set to a system monitored by a NIDS, and depending on the Path-MTU to the intended recipient, the packet might be dropped by some intervening router (being too big to be forwarded without fragmentation), without the NIDS being aware.

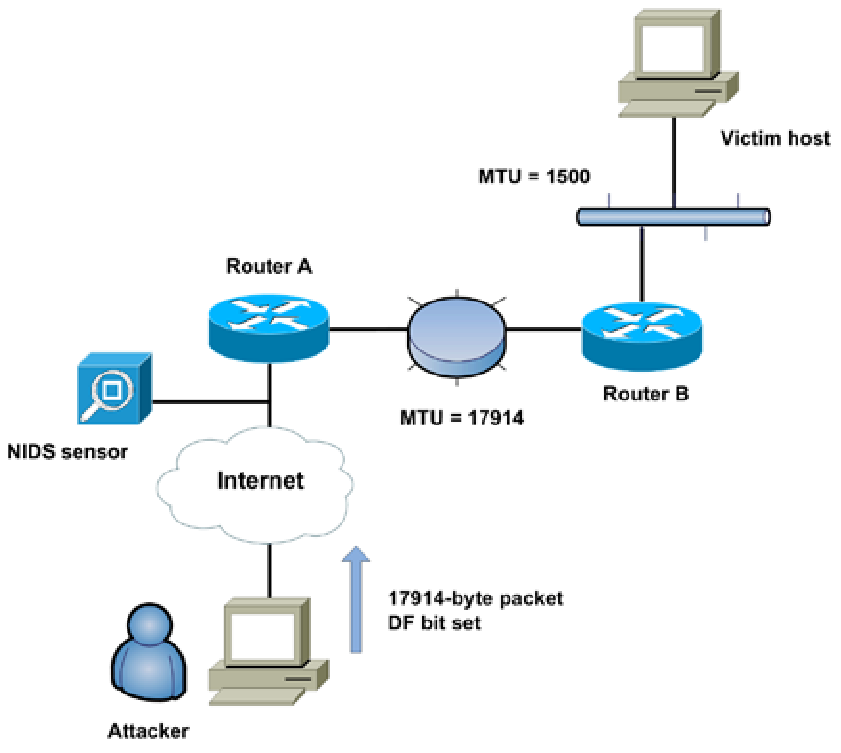

Figure 3: NIDS evasion by means of the Internet Protocol DF bit

In Figure 3, an attacker sends a 17914-byte datagram meant to the victim host. The attacker's packet probably contains an overlapping IP fragment or an overlapping TCP segment, aiming at "confusing" the NIDS, as described in [Ptacek and Newsham, 1998]. The packet is screened by the NIDS sensor at the network perimeter, which probably reassembles IP fragments and TCP segments for the purpose of assessing the data transferred to and from the monitored systems. However as the attacker's packet should transit a link with an MTU smaller than 17914 bytes (1500 bytes in this example), the router that encounters that this packet cannot be forwarded without fragmentation (Router B) discards the packet, and sends an ICMP "fragmentation needed and DF bit set" error message to the source host. In this scenario the NIDS may remain unaware that the screened packet never reached the intended destination, and thus get an incorrect picture of the data being transferred to the monitored systems.

[Shankar and Paxson, 2003] introduces a technique named "Active Mapping" that prevents evasion of a NIDS by acquiring sufficient knowledge about the network being monitored, to assess which packets will arrive at the intended recipient, and how they will be interpreted by it

Some firewalls are known to drop packets that have both the **MF** (More Fragments) and the **DF** (Don't fragment) bits set. While such a packet might seem nonsensical, there are a number of reasons why non-malicious packets with these two bits set can be found in a network. First, they may exist as the result of some middle-box processing a packet that was too large to be forwarded without fragmentation. Instead of simply dropping the corresponding packet and sending an ICMP error message to the source host, some middle-boxes fragment the packet (copying the **DF** bit to each fragment) and also send an ICMP error message to the originating system. Second, some systems (notably Linux) set both the **MF** and the **DF** bits to implement Path-MTU Discovery (PMTUD) for UDP. These scenarios should be taken into account when configuring firewalls and/or tuning Network Intrusion Detection Systems (NIDS).

3.7 Fragment Offset

The **Fragment Offset** is used for the fragmentation and reassembly of IP datagrams. It indicates where in the original datagram the fragment belongs and is measured in units of eight bytes. As a consequence all fragments (except the last one) have to be aligned on an 8-byte boundary. Therefore if a packet has the **MF** flag set the following check should be enforced:

(**Total Length** – **IHL** * 4) % 8 == 0

If the packet does not pass this check it should be dropped.

Given that **Fragment Offset** is a 13-bit field it can hold a value of up to 8191 which would correspond to an offset 65528 bytes within the original (non-fragmented) datagram. As such, it is possible for a fragment to implicitly claim to belong to a datagram larger than 65535 bytes (the maximum size for a legitimate IP datagram). Even when the fragmentation mechanism would seem to allow fragments that could reassemble into large datagrams, the intent of the specification is to allow for the transmission of datagrams of up to 65535 bytes. Therefore, if a given fragment would reassemble into a datagram of more than 65535 bytes, the resulting datagram should be dropped. To detect such a case, the following check should be enforced on all packets for which the **Fragment Offset** contains a non-zero value:

Fragment Offset * 8 + (**Total Length** – **IHL** * 4) <= 65535

In the worst-case scenario, the reassembled datagram could have a size of up to 131043 bytes.

Such a datagram would result when the first fragment has a **Fragment Offset** of 0 and a **Total Length** of 65532, and the second (and last) fragment has a **Fragment Offset** of 8189 (65512 bytes), and a **Total Length** of 65535. Assuming an **IHL** of 5 (i.e., a header length of 20 bytes), the reassembled datagram would be 65532 + (65535 – 20) = 131047 bytes.

The IP module should implement all the necessary measures to be able to handle such illegitimate reassembled datagrams, so as to avoid them from overflowing the buffer(s) used for the reassembly function.

[CERT, 1996c] and [Kenney, 1996] describe the exploitation of this issue to perform a Denial of Service (DoS) attack.

3.8 Time to Live (TTL)

The **Time to Live (TTL)** field has two functions: to bind the lifetime of the upper-layer packets (e.g. TCP segments) and to prevent packets from looping indefinitely in the network.

Originally, this field was meant to indicate maximum time a datagram was allowed to remain within the internet system in units of seconds. As every internet module that processes a datagram must decrement the **TTL** by at least one, the original definition of the **TTL** field became obsolete, and it must now be interpreted as a hop count.

Most systems allow the administrator to configure the TTL to be used for the packets sent with the default value usually being a power of 2. The recommended value for the TTL field, as specified by the IANA is 64 [IANA, 2006b]. This value reflects the assumed “diameter” of the Internet plus a margin to accommodate its growth.

The **TTL** field has a number of properties that are interesting from a security point of view. Given that the default value used for the **TTL** is usually a power of eight the chances are that, unless the originating system has been explicitly tuned to use a non-default value, if a packet arrives with a **TTL** of 60 the packet was originally sent with a **TTL** of 64. In the same way if a packet is received with a **TTL** of 120 chances are that the original packet had a **TTL** of 128.

This discussion assumes there was no protocol scrubber, transparent proxy, or some other middle-box that overwrites the **TTL** field in a non-standard way, between the originating system and the point of the network in which the packet was received.

Asserting the **TTL** with which a packet was originally sent by the source system can help to obtain valuable information. Among other things, it may help in:

- Fingerprinting the operating system being used by the source host.
- Fingerprinting the physical device from which the packets originate.
- Locating the source host in the network topology.

Additionally, it can be used to perform functions such as:

- Evading Network Intrusion Detection Systems.
- Improving the security of applications that make use of the IP.

Fingerprinting the operating system in use by the source host

Different operating systems use a different default **TTL** for the packets they send so asserting the **TTL** with which a packet was originally sent will help to reduce the number of possible operating systems in use by the source host.

Fingerprinting the physical device from which the packets originate

When several systems are behind a middle-box such as a NAT or a load balancer, the **TTL** may help to count the number of systems behind the middle-box. If each of the systems behind the middle-box use a different default **TTL** for the packets they send, or they are located in a different place of the network topology, an attacker could stimulate responses from the devices being fingerprinted and each response that arrives with a different **TTL** could be assumed to come from a different device.

Of course, there are other more precise techniques to fingerprint physical devices. Amongst the drawbacks of this method, while many systems differ in the default **TTL** they use for the packets they send, there are also many implementations which use the same default **TTL**. Additionally, packets sent by a given device may take different routes (e.g. due to load sharing or route changes) and thus a given packet may incorrectly be presumed to come from a different device when in fact it just traveled a different route.

Locating the source host in the network topology

The **TTL** field may also be used to locate the source system in the network topology [Northcutt and Novak, 2000].

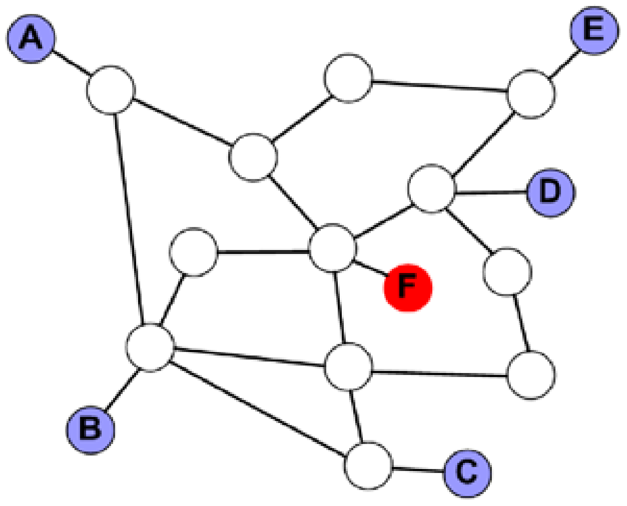

Figure 4: Tracking a host by means of the TTL field

Consider network topology of Figure 4. Assuming that an attacker ("F" in the figure) is performing an attack that requires forging the **Source Address** (such as a TCP-based DoS reflection attack), and some of the involved hosts are willing to cooperate to locate the attacking system.

Assuming that:

- All the packets A gets have a **TTL** of 61.
- All the packets B gets have a **TTL** of 61.
- All the packets C gets have a **TTL** of 61.
- All the packets D gets have a **TTL** of 62.

Based on this information, and assuming that the system's default value was not overridden, it would be fair to assume that the original **TTL** of the packets was 64. With this information the number of hops between the attacker and each of the aforementioned hosts can be calculated.

The attacker is:

- Three hops away from A.
- Three hops away from B.
- Three hops away from C.
- Two hops away from D.

In the network setup of Figure 3, the only system that satisfies all these conditions is the one marked as the "F".

The scenario described above is for illustration purposes only. In practice, there are a number of factors that may prevent this technique from being successfully applied:

- Unless there is a “large” number of cooperating systems, and the attacker is assumed to be no more than a few hops away from these systems, the number of “candidate” hosts will usually be too large for the information to be useful.
- The attacker may be using a non-default **TTL** value or, worse, using a pseudo-random value for the **TTL** of the packets it sends.
- The packets sent by the attacker may take different routes, as a result of a change in network topology, load sharing, etc., and may lead to an incorrect analysis.

Evading Network Intrusion Detection Systems

The **TTL** field can be used to evade Network Intrusion Detection Systems. Depending on the position of a sensor relative to the destination host of the examined packet, the NIDS may get a different picture from that of the intended destination system. As an example, a sensor may process a packet that will expire before getting to the destination host. A general counter-measure for this type of attack is to normalise the traffic that gets to an organisational network. Examples of such traffic normalisation can be found in [Paxson et al, 2001].

Improving the security of applications that make use of the Internet Protocol (IP)

In some scenarios the **TTL** field can be also used to improve the security of an application by restricting the hosts that can communicate with the given application. For example, there are applications for which the communicating systems are typically in the same network segment (i.e., there are no intervening routers). Such an application is the BGP (Border Gateway Protocol) between utilised by two peer routers.

If both systems use a **TTL** of 255 for all the packets they send to each other, then a check could be enforced to require all packets meant for the application in question to have a **TTL** of 255.

As all packets sent by systems that are not in the same network segment will have a **TTL** smaller than 255, those packets will not pass the check enforced by these two cooperating peers. This check reduces the set of systems that may perform attacks against the protected application (BGP in this case), thus mitigating the attack vectors described in [NISCC, 2004] and [Watson, 2004].

This same check is enforced for related ICMP error messages, with the intent of mitigating the attack vectors described in [NISCC, 2005] and [Gont, 2008].

.

The **TTL** field can similarly be used in scenarios in which the cooperating systems either do not use a default **TTL** of 255, or are not in the same network segment (i.e., multi-hop peering). In that case, the following check could be enforced:

$$\textbf{TTL} >= 255 - \Delta\text{hops}$$

This means that the set of hosts from which packets will be accepted for the protected application will be reduced to those that are no more than Δhops away. While for obvious reasons the level of protection will be smaller than in the case of directly-connected peers, the use of the **TTL** field for protecting multi-hop peering still reduces the set of hosts that could potentially perform a number of attacks against the protected application.

This use of the **TTL** field has been officially documented by the IETF under the name "Generalised TTL Security Mechanism" (GTSM) in [Gill et al, 2007].

Some protocol scrubbers enforce a minimum value for the **TTL** field of the packets they forward. It must be understood that depending on the minimum **TTL** being enforced, and depending on the particular network setup, the protocol scrubber may actually help attackers to fool the GTSM, by "raising" the **TTL** of the attacking packets.

3.9 Protocol

The **Protocol** field indicates the protocol encapsulated in the internet datagram. The **Protocol** field may not only contain a value corresponding to an implemented protocol within the system but also a value corresponding to a protocol not implemented or even a value not yet assigned by the IANA [IANA, 2006c].

While in theory there should not be security implications from the use of any value in the protocol field, there have been security issues in the past with systems that had problems when handling packets with some specific protocol numbers [Cisco, 2003] [CERT, 2003].

3.10 Header Checksum

The **Header Checksum** field is an error detection mechanism meant to detect errors in the IP header. While in principle there should not be security implications arising from this field, it should be noted that due to non-RFC-compliant implementations, the **Header Checksum** might be exploited to detect firewalls and/or evade network intrusion detection systems (NIDS).

[Ed3f, 2002] describes the exploitation of the TCP checksum for performing such actions. As there are internet routers known to not check the IP **Header Checksum**, and there might also be middle-boxes (NATs, firewalls, etc.) not checking the IP checksum allegedly due to performance reasons, similar malicious activity to the one described in [Ed3f, 2002] might be performed with the IP checksum.

3.11 Source Address

The **Source Address** of an IP datagram identifies the node from which the packet originated.

Strictly speaking, the **Source Address** of an IP datagram identifies the interface of the sending system from which the packet was sent, (rather than the originating "system"), as in the Internet Architecture there's no concept of "node".

Unfortunately it is trivial to forge the **Source Address** of an Internet datagram. This has been exploited in the past for performing a variety of DoS (Denial of Service) attacks (see [NISCC, 2004] [Eddy, 2007] [CERT, 1996a] [CERT, 1996b] [CERT, 1998a]) and to impersonate as other systems in scenarios in which authentication was based on the **Source Address** of the sending system [daemon9 et al, 1996].

The extent to which these attacks can be successfully performed in the Internet can be reduced through deployment of ingress/egress filtering in the internet routers. [NISCC, 2006] is a detailed guide on ingress and egress filtering. [Baker and Savola, 2004] and [Ferguson and Senie, 2000] discuss ingress filtering. [GIAC, 2000] discusses egress filtering.

> Even when the obvious field on which to perform checks for ingress/egress filtering is the **Source Address** and **Destination Address** fields of the IP header, there are other occurrences of IP addresses on which the same type of checks should be performed. One example is the IP addresses contained in the payload of ICMP error messages, as discussed in [Gont, 2008] and [Gont, 2006].

There are a number of sanity checks that should be performed on the **Source Address** of an IP datagram. Details can be found in Section 4.2 (“Addressing”).

Additionally, there are freely available tools that allow administrators to monitor which IP addresses are used with which MAC addresses [LBNL/NRG, 2006]. This functionality is also included in many Network Intrusion Detection Systems (NIDS).

It is also very important to understand that authentication should never rely on the **Source Address** of the communicating systems.

3.12 Destination Address

The **Destination Address** of an IP datagram identifies the destination host to which the packet is meant to be delivered.

> Strictly speaking, the **Destination Address** of an IP datagram identifies the interface of the destination network interface, rather than the destination “system”. As in the Internet Architecture there is no concept of “node”.

There are a number of sanity checks that should be performed on the **Destination Address** of an IP datagram. Details can be found in Section 4.2 (“Addressing”).

3.13 Options

According to RFC 791, IP options must be implemented by all IP modules, both in hosts and gateways (i.e., end-systems and intermediate-systems).

There are two cases for the format of an option:

- Case 1: A single byte of **option-type**.
- Case 2: An **option-type** byte, an **option-length** byte, and the actual **option-data** bytes.

In Case 2, the **option-length** byte counts the **option-type** byte and the **option-length** byte, as well as the actual option-data bytes.

All options, except “End of Option List” (Type = 0) and “No Operation” (Type = 1), are of Class 2.

The **option-type** has three fields:

- 1 bit: **copied flag**.
- 2 bits: **option class**.
- 5 bits: **option number**.

The **copied flag** indicates whether this option should be copied to all fragments when the packet carrying it needs to be fragmented:

- 0 = not copied.
- 1 = copied.

The values for the **option class** are:

- 0 = control.
- 1 = reserved for future use.
- 2 = debugging and measurement.
- 3 = reserved for future use.

This format allows for the creation of new options for the extension of the IP.

Finally, the **option number** identifies the syntax of the rest of the option.

3.13.1 General issues with IP options

The following subsections discuss security issues that apply to all IP options. The proposed checks should be performed in addition to any option-specific checks proposed in the next sections.

3.13.1.1 Processing requirements

Router manufacturers tend to do IP option processing on the main processor rather than on line cards. Unless special care is taken this may be a security risk as there is potential for overwhelming the router with option processing.

To reduce the impact of these packets on the system performance, a few counter-measures could be implemented:

- Rate-limit the number of packets with IP options that are processed by the system.
- Enforce a limit on the maximum number of options to be accepted on a given internet datagram.

The first check avoids a flow of packets with IP options to overwhelm the system in question. The second check avoids packets with multiple IP options to affect the performance of the system.

3.13.1.2 Processing of the options by the upper layer protocol

Section 3.2.1.8 of RFC 1122 [Braden, 1989] states that all the IP options received in IP datagrams must be passed to the transport layer (or to ICMP processing when the datagram is an ICMP message). Care in option processing must be taken not only at the internet layer but also in every protocol module that may end up processing the options included in an IP datagram.

3.13.1.3 General sanity checks on IP options

There are a number of sanity checks that should be performed on IP options before further option processing is done. They help prevent a number of potential security problems including buffer overflows. When these checks fail the packet carrying the option should be dropped.

RFC 1122 [Braden, 1989] recommends to send an ICMP "Parameter Problem" message to the originating system when a packet is dropped because of a invalid value in a field, such as the cases discussed in the following subsections. Sending such a message might help in debugging some network problems. However it would also alert attackers about the system that is dropping packets because of the invalid values in the protocol fields.

We advise that systems default to sending an ICMP "Parameter Problem" error message when a packet is dropped because of an invalid value in a protocol field (e.g. as a result of dropping a packet due to the sanity checks described in this section). However we recommend that systems provide a system-wide toggle that allows an administrator to override the default behavior so that packets can be silently dropped when an invalid value in a protocol field is encountered.

Option length

Section 3.2.1.8 of RFC 1122 explicitly states that the IP layer must not crash as the result of an option length that is outside the possible range, and mentions that erroneous option lengths have been observed to put some IP implementations into infinite loops.

For options that belong to the "Case 2" described in the previous section, the following check should be performed:

option-length >= 2

The value “2” accounts for the **option-type** byte, and the **option-length** byte.

This check prevents, among other things, loops in option processing that may arise from incorrect option lengths.

Additionally, while the **option-length** byte of IP options of “Case 2” allows for an option length of up to 255 bytes, there is a limit on legitimate option length imposed by the syntax of the IP header.

For all options of “Case 2”, the following check should be enforced:

option-offset + **option-length** <= **IHL** * 4

Where option-offset is the offset of the first byte of the option within the IP header, with the first byte of the IP header being assigned an offset of 0.

If a packet does not pass both of these checks, it should be dropped.

The aforementioned check is meant to detect forged **option-length** values that might make an option overlap with the IP payload. This would be particularly dangerous for those **IP options** which request the processing systems to write information into the option-data area (such as the Record Route option), as it would allow the generation of overflows.

Data types

Many IP options use pointer and length fields. Care must be taken as to the data type used for these fields in the implementation. For example, if an 8-bit signed data type were used to hold an 8-bit pointer, then pointer values larger than 128 might mistakenly be interpreted as negative numbers and might lead to unpredictable results.

3.13.2 Issues with specific options

3.13.2.1 End of Option List (Type = 0)

This option is used to indicate the “end of options” in those cases in which the end of options would not coincide with the end of the Internet Protocol Header.

IP systems are required to ignore those options they do not implement. Therefore, even in those cases in which this option is required, but is missing, IP systems should be able to process the remaining bytes of the IP header without any problems.

3.13.2.2 No Operation (Type = 1)

The no-operation option is basically meant to allow the sending system to align subsequent options in, for example, 32-bit boundaries.

This option does not have security implications.

3.13.2.3 Loose Source Record Route (LSRR) (Type = 131)

This option lets the originating system specify a number of intermediate systems a packet must pass through to get to the destination host. Additionally, the route followed by the packet is recorded in the option. The receiving host (end-system) must use the reverse of the path contained in the received LSRR option.

The LSSR option can be of help in debugging some network problems. Some ISP Internet Service Provider) peering agreements require support for this option in the routers within the peer of the ISP.

The LSRR option has well-known security implications. Among other things, the option can be used to:

- Bypass firewall rules
- Reach otherwise unreachable internet systems
- Establish TCP connections in a stealthy way
- Learn about the topology of a network
- Perform bandwidth-exhaustion attacks

Of these attack vectors, the one that has probably received least attention is the use of the LSRR option to perform bandwidth exhaustion attacks. The LSRR option can be used as an amplification method for bandwidth-exhaustion attacks as an attacker could make a packet bounce multiple times between a number of systems by carefully crafting an LSRR option.

> This is the IPv4-version of the IPv6 amplification attack that was widely publicised in 2007 [Biondi and Ebalard, 2007]. The only difference is that the maximum length of the IPv4 header (and hence the LSRR option) limits the amplification factor when compared to the IPv6 counter-part.

While the LSSR option may be of help in debugging some network problems, its security implications outweigh any legitimate use.

All systems should, by default, drop IP packets that contain an LSRR option. However,they should provide a system-wide toggle to enable support for this option for those scenarios where option is required. Such a system-wide toggle should default to “off” (or “disable”).

[OpenBSD, 1998] is a security advisory about an improper implementation of such a system-wide toggle in 4.4BSD kernels.

Section 3.3.5 of RFC 1122 [Braden, 1989] states that a host may be able to act as an intermediate hop in a source route, forwarding a source-routed datagram to the next specified hop. We strongly discourage host software from forwarding source-routed datagrams.

If processing of source-routed datagrams is explicitly enabled in a system, the following sanity checks should be performed.

RFC 791 states that this option should appear, at most, once in a given packet.If a packet is found to have more than one LSRR option it should be dropped. Therefore, hosts and routers should discard packets that contain more than one LSRR option. Additionally, if a packet were found to have both LSRR and SSRR options it should be dropped.

As many other IP options the LSSR contains a Length field that indicates the length of the option. Given the format of the option the Length field should be checked to be at least 3 (three):

LSRR.Length >= 3

If the packet does not pass this check it should be dropped.

Additionally, the following check should be performed on the **Length** field:

LSRR.Offset + **LSRR.Length** < **IHL** *4

This check assures that the option does not overlap with the IP payload (i.e., it does not go past the IP header). If the packet does not pass this check it should be dropped.

The **Pointer** is relative to this option. Thus, the minimum legal value is 4. Therefore, the following check should be performed.

LSRR.Pointer >= 4

If the packet does not pass this check it should be dropped. Additionally, the **Pointer** field should be a multiple of 4. Consequently, the following check should be performed:

LSRR.Pointer % 4 == 0

If a packet does not pass this check it should be dropped.

When a system receives an IP packet with the LSRR route option, it should check whether the source route is empty or not. The option is empty if:

LSRR.Pointer > **LSRR.Length**

In that case routing should be based on the **Destination Address** field and no further processing should be done on the LSRR option.

[Microsoft, 1999] is a security advisory about a vulnerability arising from improper validation of the **LSRR.Pointer** field.

If the address in the **Destination Address** field has been reached, and the option is not empty, the next address in the source route replaces the address in the **Destination Address** field.

The IP address of the interface that will be used to forward this datagram should be recorded into the LSRR. However, before writing in the **route data** area, the following check should be performed:

LSRR.Length – LSRR.Pointer >= 3

This assures that there will be at least 4 bytes of space in which to record the IP address. If the packet does not pass this check it should be dropped.

An offset of "1" corresponds to the option type, that's why the performed check is LSRR.Length – LSRR.Pointer >=3, and not LSRR.Length – LSRR.Pointer >=4.

The LSRR must be copied on fragmentation. This means that if a packet that carries the LSRR is fragmented, each of the fragments will have to go through the list of systems specified in the LSRR option.

3.13.2.4 Strict Source and Record Route (SSRR) (Type = 137)

This option allows the originating system to specify a number of intermediate systems a packet must pass through to get to the destination host. Additionally, the route followed by the packet is recorded in the option, and the destination host (end-system) must use the reverse of the path contained in the received SSRR option.

This is similar to the LSRR option with the only difference that in the case of SSRR the route specified in the option is the exact route the packet must take (i.e., no other intervening routers are allowed to be in the route).

The SSSR option can be of help in debugging some network problems. Some ISP Internet Service Provider) peering agreements require support for this option in the routers within the peer of the ISP.

The SSRR option has well-known security implications. Among other things, the option can be used to:

- Bypass firewall rules
- Reach otherwise unreachable internet systems
- Establish TCP connections in a stealthy way
- Learn about the topology of a network
- Perform bandwidth-exhaustion attacks

Of these attack vectors, the one that has probably received least attention is the use of the SSRR option to perform bandwidth exhaustion attacks. The SSRR option can be used as an amplification method for bandwidth-exhaustion attacks as an attacker could make a packet bounce multiple times between a number of systems by carefully crafting an LSRR option.

> This is the IPv4-version of the IPv6 amplification attack that was widely publicised in 2007 [Biondi and Ebalard, 2007]. The only difference is that the maximum length for the IPv4 header (and hence the SSRR option) limits the amplification factor when compared to the IPv6 counter-part.

While the SSSR option may be of help in debugging some network problems its security implications outweigh any legitimate use of it.

All systems should, by default, drop IP packets that contain an LSRR option. However, they should provide a system-wide toggle to enable support for this option for those scenarios in which this option is required. Such system-wide toggle should default to “off” (or “disable”).

> [OpenBSD, 1998] is a security advisory about an improper implementation of such a system-wide toggle in 4.4BSD kernels.

Should processing of the SSRR option be explicitly enabled there are some sanity checks that should be performed.

RFC 791 states that this option should appear, at most, once in a given packet. Thus, if a packet is found to have more than one SSRR option it should be dropped. Also, if a packet contains a combination of SSRR and LSRR options it should be dropped.

As the SSRR option is meant to specify the route a packet should follow from source to destination, use of more than one SSRR option in a single packet would be nonsensical. Therefore hosts and routers should check the IP header and discard the packet if it contains more than one SSRR option or a combination of LSRR and SSRR options.

As with many other IP options the SSRR option contains a **Length** field that indicates the length of the option. The length field should be checked to be at least 3:

SSRR.Length >= 3

If the packet does not pass this check it should be dropped.

Additionally, the following check should be performed on the length field:

SSRR.Offset + **SSRR.Length** < **IHL** *4

This check assures that the option does not overlap with the IP payload (i.e., it does not go past the IP header). If the packet does not pass this check it should be dropped.

The **Pointer** field is relative to this option, with the minimum legal value being 4 and so the following check should be performed:

SSRR.Pointer >= 4

If the packet does not pass this check it should be dropped.

Additionally, the **Pointer** field should be a multiple of 4. Consequently, the following check should be performed:

SSRR.Pointer % 4 == 0

If a packet does not pass this check it should be dropped.

If the packet passes the above checks the receiving system should determine whether the **Destination Address** of the packet corresponds to one of its IP addresses. If does not, it should be dropped.

Contrary to the IP Loose Source and Record Route (LSRR) option, the SSRR option does not allow in the route other routers than those contained in the option. If the system implements the weak end-system model it is allowed for the system to receive a packet destined to any of its IP addresses on any of its interfaces. If the system implements the strong end-system model a packet destined to it can be received only on the interface that corresponds to the IP address contained in the **Destination Address** field [Braden, 1989].

If the packet passes this check the receiving system should determine whether the source route is empty or not. The option is empty if:

SSRR.Pointer > **SSRR.Length**

In that case, if the address in the destination field has not been reached the packet should be dropped.

[Microsoft, 1999] is a security advisory about a vulnerability arising from improper validation of the **SSRR.Pointer** field.

If the option is not empty, and the address in the **Destination Address** field has been reached, the next address in the source route replaces the address in the **Destination Address** field. This IP address must be reachable without the use of any intervening router (i.e., the address must belong to any of the networks to which the system is directly attached). If that is not the case the packet should be dropped.

The IP address of the interface that will be used to forward this datagram should be recorded into the SSRR. However, before doing that, the following check should be performed:

SSRR.Length – SSRR.Pointer >=3

An offset of "1" corresponds to the option type, that's why the performed check is **SSRR.Length – SSRR.Pointer** >=3, and not **SSRR.Length – SSRR.Pointer** >=4.

This assures that there will be at least 4 bytes of space on which to record the IP address. If the packet does not pass this check it should be dropped.

The SSRR option must be copied on fragmentation. This means that if a packet that carries the SSRR is fragmented, each of the fragments will have to go through the list of systems specified in the SSRR option.

3.13.2.5 Record Route (Type = 7)

This option provides a means to record the route that a given packet follows.

The option begins with an 8-bit option code which must be equal to 7. The second byte is the option length, which includes the **option-type** byte, the **option-length** byte, the **pointer** byte, and the actual **option-data.** The third byte is a pointer into the route data, indicating the first byte of the area in which to store the next route data. The pointer is relative to the option start.

RFC 791 states that this option should appear once, at most, in a given packet. Therefore, if a packet has more than one instance of this option it should be dropped.

Given the format of the option, the **Length** field should be checked to be at least 3:

RR.Length >= 3

If the packet does not pass this check it should be dropped.

Additionally, the following check should be performed on the **Length** field:

RR.Offset + **RR_Length** < **IHL** *4

This check assures that the option does not overlap with the IP payload (i.e., it does not go past the IP header). If the packet does not pass this check it should be dropped.

The pointer field is relative to this option, with the minimum legal value being 4. Therefore the following check should be performed:

RR.Pointer >= 4

If the packet does not pass this check it should be silently dropped.

Additionally, the **Pointer** field should be a multiple of 4. Consequently, the following check should be performed:

RR.Pointer % 4 == 0

If the packet does not pass this check it should be dropped.

When a system receives an IP packet with the Record Route option, it should check whether there is space in the option to store route information. The option is full if:

RR.Pointer > **RR.Length**

If the option is full, the datagram should be forwarded without further processing of this option. If not, the following check should be performed before writing any route data into the option:

RR.Length - RR.Pointer >= 3

An offset of "1" corresponds to the option type, that's why the performed check is **LSRR.Length – LSRR.Pointer** >=3, and not **LSRR.Length – LSRR.Pointer** >=4.

If the packet does not pass this check the packet should be considered in error and therefore should be silently dropped.

> If the option is not full (i.e., **RR.Pointer** <= **RR.Length**), but **RR.Length – RR.Pointer** < 3, it means that while there's space in the option, there is not enough space to store an IP address. It is fair to assume that such a scenario will only occur when the packet has been crafted.

If the packet passes this check the IP address of the interface that will be used to forward this datagram should be recorded into the area pointed by the **RR.Pointer**, and **RR.Pointer** should then be incremented by 4.

This option is not copied on fragmentation and appears in the first fragment only. If a fragment other than the one with offset 0 contains the Record Route option it should be dropped.

The Record Route option can be exploited to learn about the topology of a network, as it allows an attacker to record.

3.13.2.6 Stream Identifier (Type = 136)

The Stream Identifier option originally provided a means for the 16-bit SATNET stream Identifier to be carried through networks that did not support the stream concept.

However, as stated by Section 4.2.2.1 of RFC 1812 [Baker, 1995], this option is obsolete. Therefore it should be ignored by the processing systems.

In the case of legacy systems still using this option, the length field of the option should be checked to be 4. If the option does not pass this check it should be dropped.

RFC 791 states that this option appears at most once in a given datagram. If a packet contains more than one instance of this option it should be dropped.

3.13.2.7 Internet Timestamp (Type = 68)

This option provides a means for recording the time at which each system processed this datagram. The timestamp option has a number of security implications such as:

- It allows an attacker to obtain the current time of the systems that process the packet, which the attacker may find useful in a number of scenarios.
- It may be used to map the network topology, in a similar way to the IP Record Route option.
- It may be used to fingerprint the operating system in use by a system processing the datagram.
- It may be used to fingerprint physical devices, by analysing the clock skew.

Therefore, by default, the timestamp option should be ignored.

For those systems that have been explicitly configured to honor this option, the rest of this subsection describes some sanity checks that should be enforced on the option before further processing.

The option begins with an **option-type** byte which must be equal to 68. The second byte is the **option-length** which includes the **option-type** byte, the **option-length** byte, the **pointer** and the **overflow/flag** byte. The minimum legal value for the option-length byte is 4, which corresponds to an Internet Timestamp option that is empty (no space to store timestamps). Therefore upon receipt of a packet that contains an Internet Timestamp option the following check should be performed:

IT.Length >= 4

If the packet does not pass this check it should be dropped.

The following check should also be performed on the option length field:

IT.Offset + **IT.Length** < **IHL** *4

This check assures that the option does not overlap with the IP payload (i.e., it does not go past the IP header). If the packet does not pass this check it should be dropped.

The **pointer** byte points to the first byte of the area in which the next timestamp data should be stored. As its value is relative to the beginning of the option its minimum legal value is 5. Consequently, the following check should be performed on a packet that contains the Internet Timestamp option:

IT.Pointer >= 5

If the packet does not pass this check it should be dropped.

The **flag** field has three possible legal values:

- 0: Record time stamps only, stored in consecutive 32-bit words.
- 1: Record each timestamp preceded with the internet address of the registering entity.
- 3: The internet address fields of the option are pre-specified. An IP module only registers its timestamp if it matches its own address with the next specified internet address.

Therefore the following check should be performed:

IT.Flag == 0 || **IT.Flag** == 1 || **IT.Flag** == 3

If the packet does not pass this check it should be dropped.

The **timestamp** field is a right-justified 32-bit timestamp in milliseconds since UT. If the time is not available in milliseconds, or cannot be provided with respect to UT, then any time may be inserted as a timestamp, provided the high order bit of the timestamp is set, to indicate this non-standard value.

According to RFC 791, the initial contents of the timestamp area must be initialised to zero, or internet address/zero pairs. However, internet systems should be able to handle non-zero values possibly discarding the offending datagram.

When an internet system receives a packet with an Internet Timestamp option it decides whether it should record its timestamp in the option. If so,, it should determine whether the timestamp data area is full by means of the following check:

IT.Pointer > **IT.Length**

If this condition is true the timestamp data area is full. If not, there is room in the timestamp data area.

If the timestamp data area is full, the **overflow** byte should be incremented, and the packet should be forwarded without inserting the timestamp. If the **overflow** byte itself overflows the packet should be dropped.

If timestamp data area is not full further checks should be performed before actually inserting any data.

- If the **IT.Flag** byte is 0 the following check should be performed:

IT.Length – IT.Pointer >= 3

If the packet does not pass this check it should be dropped. If the packet passes this check there is room for at least one 32-bit timestamp. The system's 32-bit timestamp should be inserted at the area pointed by the pointer byte, and the pointer byte should be incremented by four.

- If the **IT.Flag byte** is 1 then the following check should be performed:

IT.Length – IT.Pointer >= 7

If the packet does not pass this check it should be dropped. If the packet does pass this check it means there is space in the timestamp data area to store at least one IP address plus the corresponding 32-bit timestamp. The IP address of the system should be stored at the area pointed to by the pointer byte, followed by the 32-bit system timestamp. The pointer byte should then be incremented by 8.

- If the flag byte is 3 then the following check should be performed:

IT.Length – IT.Pointer >= 7

If the packet does not pass this check it should be dropped. If it does it means there is space in the timestamp data area to store an IP address and store the corresponding 32-bit timestamp. The system's timestamp should be stored at the area pointed by **IT.Pointer** + 4. The pointer byte should be incremented by 8.

[Kohno et al, 2005] describes a technique for fingerprinting devices by measuring the clock skew. It exploits, among other things, the timestamps that can be obtained by means of the ICMP timestamp request messages [Postel, 1981]. The same fingerprinting method could be implemented with the aid of the Internet Timestamp option.

3.13.2.8 Router Alert (Type = 148)

The Router Alert option is defined in RFC 2113 [Katz, 1997]. It has the semantic “routers should examine this packet more closely”. A packet that contains a Router Alert option will not go through the router’s fast-path and will be processed in the router more slowly than if the option were not set. Therefore this option may impact the performance of the systems that handle the packet carrying it.

According to the syntax of the option as defined in RFC 2113, the following check should be enforced:

RA.Length == 4

If the packet does not pass this check it should be dropped. Furthermore, the following check should be performed on the **Value** field:

RA.Value == 0

If the packet does not pass this check it should be dropped.

As explained in RFC 2113 hosts should ignore this option.

3.13.2.9 Probe MTU (Type =11)

This option is defined in RFC 1063 [Mogul et al, 1988] and originally provided a mechanism to discover the Path-MTU.

This option is obsolete and therefore any packet that is received containing this option should be dropped.

3.13.2.10 Reply MTU (Type = 12)

This option is defined in RFC 1063 [Mogul et al, 1988] and originally provided a mechanism to discover the Path-MTU.

This option is obsolete and therefore any packet that is received containing this option should be dropped.

3.13.2.11 Traceroute (Type = 82)

This option is defined in RFC 1393 [Malkin, 1993], and originally provided a mechanism to trace the path to a host.

This option is obsolete, and therefore any packet that is received containing this option should be dropped.

3.13.2.12 DoD Basic Security Option (Type = 130)

This option is used by end-systems and intermediate systems of an internet to [Kent, 1991]:

- Transmit from source to destination in a network standard representation the common security labels required by computer security models.
- Validate the datagram as appropriate for transmission from the source and delivery to the destination.
- Ensure that the route taken by the datagram is protected to the level required by all protection authorities indicated on the datagram.

It is specified by RFC 1108 [Kent, 1991] (which obsoletes RFC 1038 [St. Johns, 1988]).

RFC 791 [Postel, 1981] defined the "Security Option" (Type = 130), which used the same option type as the DoD Basic Security option discussed in this section. The "Security Option" specified in RFC 791 is considered obsolete by Section 4.2.2.1 of RFC 1812, and therefore the discussion in this section is focused on the DoD Basic Security option specified by RFC 1108 [Kent, 1991].

Section 4.2.2.1 of RFC 1812 states that routers "SHOULD implement this option".

The DoD Basic Security Option is currently implemented in a number of operating systems (e.g. [IRIX, 2008], [SELinux, 2008], [Solaris, 2008], and [Cisco, 2008]), and deployed in a number of high-security networks.

RFC 1108 states that the option should appear once in a datagram at the most. If more than one DoD Basic Security option (BSO) appears in a given datagram the corresponding datagram should be dropped.

RFC 1108 states that the option **Length** is variable with a minimum option **Length** of 3 bytes. Therefore the following check should be performed:

BSO.Length >= 3

If the packet does not pass this check it should be dropped.

Systems that belong to networks in which this option is in use should process the DoD Basic Security option contained in each packet as specified in [Kent, 1991].

Current deployments of the DoD Security Options have motivated the proposal of a "Common Architecture Label IPv6 Security Option (CALIPSO)" for the IPv6 protocol. [St. Johns et al, 2008].

3.13.2.13 DoD Extended Security Option (Type = 133)

This option permits additional security labeling information, beyond that present in the Basic Security Option (Section 3.13.2.12), to be supplied in an IP datagram to meet the needs of registered authorities. It is specified by RFC 1108 [Kent, 1991].

This option may be present only in conjunction with the DoD Basic Security option. Therefore if a packet contains a DoD Extended Security option (ESO), but does not contain a DoD Basic Security option, it should be dropped. It should be noted that,unlike the DoD Basic Security option, this option may appear multiple times in a single IP header.

RFC 1108 states that the option Length is variable, with a minimum option Length of 3 bytes. Therefore the following check should be performed:

ESO.Length >= 3

If the packet does not pass this check it should be dropped.

Systems that belong to networks in which this option is in use should process the DoD Extended Security option contained in each packet as specified in RFC 1108 [Kent, 1991].

3.13.2.14 Commercial IP Security Option (CIPSO) (Type = 134)

This option was proposed by the Trusted Systems Interoperability Group (TSIG) with the intent of meeting trusted networking requirements for the commercial trusted systems market place. It is specified in [CIPSO, 1992] and [FIPS, 1994].

The TSIG proposal was taken to the Commercial Internet Security Option (CIPSO) Working Group of the IETF [CIPSOWG, 1994], and an Internet-Draft was produced [CIPSO, 1992]. The Internet-Draft was never published as an RFC, and the proposal was later standardised by the U.S. National Institute of Standards and Technology (NIST) as "Federal Information Processing Standard Publication 188" [FIPS, 1994].

It is currently implemented in a number of operating systems (e.g. IRIX [IRIX, 2008], Security-Enhanced Linux [SELinux, 2008], and Solaris [Solaris, 2008]) and deployed in a number of high-security networks.

[Zakrzewski and Haddad, 2002] and [Haddad and Zakrzewski, 2004] provide an overview of a Linux implementation.

According to the option syntax specified in [CIPSO, 1992] the following validation check should be performed:

CIPSO.Length >= 6

If a packet does not pass this check it should be dropped.

Systems that belong to networks in which this option is in use should process the CIPSO option contained in each packet as specified in [CIPSO, 1992].

3.13.2.15 Sender Directed Multi-Destination Delivery (Type = 149)

This option is defined in RFC 1770 [Graff, 1995], and originally provided unreliable UDP delivery to a set of addresses included in the option.

This option is obsolete. If a received packet contains this option it should be dropped.

3.14 Differentiated Services field

The Differentiated Services Architecture is intended to enable scalable service discrimination in the Internet without the need for per-flow state and signaling at every hop [Blake et al, 1998]. RFC 2474 [Nichols et al, 1998] defines a **Differentiated Services** Field (**DS** Field), which is intended to supersede the original **Type of Service** field. Figure 4 shows the format of the field.

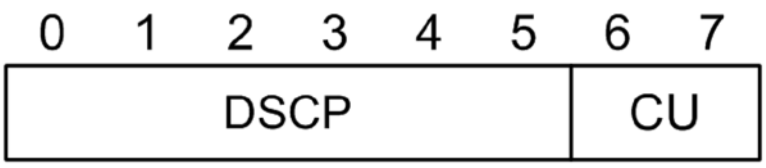

Figure 5: Structure of the DS Field

The **DSCP ("Differentiated Services CodePoint")** is used to select the treatment the packet is to receive within the Differentiated Services Domain. The **CU** ("Currently Unused") field was, at the time the specification was issued, reserved for future use. The **DSCP** field is used to select a PHB by matching against the entire 6-bit field.

Considering that the **DSCP** field determines how a packet is treated within a DS domain, an attacker sends packets with a forged **DSCP** field to perform a theft of service or even a Denial of Service attack. In particular, an attacker could forge packets with a **codepoint** of the type ‘11x000’ which, according to Section 4.2.2.2 of RFC 2474 [Nichols et al, 1998], would give the packets preferential forwarding treatment when compared with the PHB selected by the codepoint ‘000000’. If strict priority queuing were utilised, a continuous stream of such pockets could perform a Denial of Service to other flows which have a **DSCP** of lower relative order.

As the **DS** field is incompatible with the original **Type of Service** field, both DS domains and networks using the original **Type of Service** field should protect themselves by re-marking the corresponding field where appropriate, probably deploying remarking boundary nodes. Care must be taken so that packets received with an unrecognised **DSCP** do not cause the handling system to malfunction.

3.15 Explicit Congestion Notification (ECN)

RFC 3168 [Ramakrishnan et al, 2001] specifies a mechanism for routers to signal congestion to hosts sending IP packets by marking the offending packets, rather than discarding them. RFC 3168 defines the **ECN** field which utilises the **CU** unused field of the **DSCP** field described in Section 3.14 of this document. Figure 5 shows the syntax of the **ECN** field, together with the **DSCP** field used for Differentiated Services.

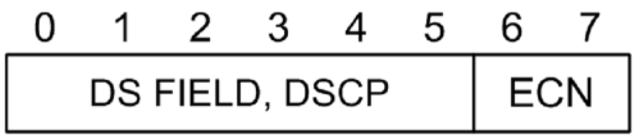

Figure 6: The Differentiated Services and ECN fields in IP

As such, the ECN field defines four codepoints:

ECN field	Codepoint
00	Not-ECT
01	ECT(1)
10	ECT(0)
11	CE

The security implications of ECN are discussed in detail in a number of Sections of RFC 3168. Of the possible threats discussed in the ECN specification we believe that one that can be easily exploited is the host falsely indicating ECN-Capability.

An attacker could set the ECT codepoint in the packets it sends to signal the network that the endpoints of the transport protocol are ECN-capable. Consequently, when experiencing moderate congestion, routers using active queue management based on

RED would mark the packets (with the CE codepoint) rather than discard them. In the same scenario, packets of competing flows that do not have the ECT codepoint set would be dropped. Therefore an attacker would get better network service than the competing flows.

However if this moderate congestion turned into heavy congestion routers should switch to drop packets, regardless of whether the packets have the ECT codepoint set or not.

A number of other threats could arise if an attacker was a man in the middle (i.e. was in the middle of the path the packets travel to get to the destination host). For a detailed discussion of those cases, we urge the reader to consult Section 16 of RFC 3168.

4. Internet Protocol Mechanisms

4.1 Fragment reassembly

To accommodate networks with different Maximum Transmission Units (MTUs) the Internet Protocol provides a mechanism for the fragmentation of IP packets by end-systems (hosts) and/or intermediate systems (routers). Reassembly of fragments is performed only by the end-systems.

[Cerf and Kahn, 1974] provides the rationale for which packet reassembly is not performed by intermediate systems.

During the last few decades IP fragmentation and reassembly has been exploited in a number of ways to perform actions such as evading Network Intrusion Detection Systems (NIDS), bypassing firewall rules, and performing Denial of Service (DoS) attacks.

[Bendi 1998] and [Humble, 1998] are examples of the exploitation of these issues for performing Denial of Service (DoS) attacks. [CERT, 1997] and [CERT, 1998b] document these issues. [Anderson, 2001] is a survey of fragmentation attacks. [US-CERT, 2001] is an example of the exploitation of IP fragmentation to bypass firewall rules. [CERT, 1999] describes the implementation of fragmentation attacks in Distributed Denial of Service (DDoS) attack tools.

The basic problem with IP fragment reassembly is the complexity of the function in a number of aspects:

- Fragment reassembly is a stateful operation for a stateless protocol (IP). The IP module at the host performing the reassembly function must allocate memory buffers both for temporarily storing the received fragments and to perform the reassembly function. Attackers can exploit this fact to exhaust memory buffers at the system performing the fragment reassembly.

- The fragmentation and reassembly mechanisms were designed at a time in which the available bandwidths were very different from the bandwidths available now. With the currently available bandwidths a number of interoperability problems may arise and these issues may be intentionally exploited by attackers to perform Denial of Service (DoS) attacks.

- Fragment reassembly must usually be performed without any knowledge of the properties of the path the fragments follow. Without this information hosts cannot make any educated guess on how long they should wait for missing fragments to arrive.

- The fragment reassembly algorithm, as described by the IETF specifications is ambiguous and allows for a number of interpretations, each of which has found place in different TCP/IP stack implementations.

- The reassembly process is somewhat complex. Fragments may arrive out of order, duplicated, overlapping, etc. and this complexity has lead to numerous bugs in different implementations of the IP protocol.

4.1.1 Problems related with memory allocation

When an IP datagram is received by an end-system it will be temporarily stored in system memory until the IP module processes it and hands it to the protocol machine that corresponds to the encapsulated protocol.

In the case of fragmented IP packets, while the IP module may perform preliminary processing of the IP header (such as checking the header for errors and processing IP options), fragments must be kept in system buffers until all fragments are received and are reassembled into a complete internet datagram.

As mentioned above, because the internet layer will not usually have information about the characteristics of the path between the system and the remote host, no educated guess can be made on the amount of time that should be waited for the other fragments to arrive. Therefore, the specifications recommend to wait for a period of time that will be acceptable for virtually all the possible network scenarios in which the protocols might operate. Specifically, RFC 1122 [Braden, 1989] states that the reassembly timeout should be a fixed value between 60 and 120 seconds. If after waiting for that period of time the remaining fragments have not yet arrived, all the received fragments for the corresponding packet are discarded.

> The original IP Specification, RFC 791 [Postel, 1981], states that systems should wait for at least 15 seconds for the missing fragments to arrive. Systems that follow the “Example Reassembly Procedure” described in Section 3.2 of RFC 791 may end up using a reassembly timer of up to 4.25 minutes, with minimum of 15 seconds. Section 3.3.2 (“Reassembly”) of RFC 1122 corrected this advice, stating that the reassembly timeout should be a fixed value between 60 and 120 seconds.

However, the longer the system waits for the missing fragments to arrive, the longer the corresponding system resources must be tied to the corresponding packet. The amount of system memory is finite and even with today’s systems it can still be considered a scarce resource.

An attacker could take advantage of the uncomfortable situation the system performing fragment reassembly is in by sending forged fragments that will never reassemble into a complete datagram. That is, an attacker would send many different fragments, with different IP IDs, without ever sending all the necessary fragments that would be needed to reassemble them into a full IP datagram. For each of the fragments the IP module would allocate resources and tie them to the corresponding fragment until the reassembly timer for the corresponding packet expires.

There are some implementation strategies which could increase the impact of this attack. For example, upon receipt of a fragment, some systems allocate a memory buffer that will be large enough to reassemble the whole datagram.

While this might be beneficial in legitimate cases, this just amplifies the impact of the possible attacks, as a single small fragment could tie up memory buffers for the size of an extremely large (and unlikely) datagram. The implementation strategy suggested in RFC 815 [Clark, 1982] leads to such an implementation.

The impact of the aforementioned attack may vary depending on some specific implementation details:

- If the system does not enforce limits on the amount of memory that can be allocated by the IP module then an attacker could tie all system memory to fragments at which point the system would become unusable, probably crashing.
- If the system enforces limits on the amount of memory that can be allocated by the IP module as a whole then when this limit is reached all further IP packets that arrive would be discarded until some fragments time out and free memory is available again.
- If the system enforces limits on the amount memory that can be allocated for the reassembly of fragments (in addition to enforcing a limit for the IP module as a whole) then when this limit is reached, all further fragments that arrive would be discarded until some fragment(s) time out and free memory is available again.

4.1.2 Problems that arise from the length of the IP Identification field

The Internet Protocols are currently being used in environments that are quite different from the ones in which they were conceived. For instance, the availability of bandwidth at the time the Internet Protocol was designed was completely different from the availability of bandwidth in today's networks.

The **Identification** field is a 16-bit field that is used for the fragmentation and reassembly function. In the event a datagram gets fragmented, all the corresponding fragments will share the same **Identification** number. Thus, the system receiving the fragments will be able to uniquely identify them as fragments that correspond to the same IP datagram. At a given point in time there must be at most only one packet in the network with a given **Identification** number. If not, an **Identification** number "collision" might occur and the receiving system might end up "mixing" fragments that correspond to different IP datagrams which simply reused the same **Identification** number.

For each group of fragments whose **Identification** numbers "collide" the fragment reassembly will lead to corrupted packets. The IP payload of the reassembled datagram will be handed to the corresponding upper layer protocol, where the error will (hopefully) be detected by some error detecting code (such as the TCP checksum) and the packet will be discarded.

An attacker could exploit this fact to intentionally cause a system to discard all or part of the fragmented traffic it gets, thus performing a Denial of Service attack. Such an attacker would simply establish a flow of fragments with different IP Identification numbers to trash all or part of the IP **Identification** space. As a result, the receiving system would use the attacker's fragments for the reassembly of legitimate datagrams, leading to corrupted packets which would later (and hopefully) get dropped.

In most cases, use of a long fragment timeout will benefit the attacker as forged fragments will keep the IP Identification space trashed for a longer period of time.

4.1.3 Problems that arise from the complexity of the reassembly algorithm

As IP packets can be duplicated by the network, and each packet may take a different path to get to the destination host, fragments may arrive not only out of order and/or duplicated, but also overlapping. This means that the reassembly process can be somewhat complex with the corresponding implementation being not specifically trivial.

[Shannon et al, 2001] analyses the causes and attributes of fragment traffic measured in several types of WANs.

During the years, a number of attacks have exploited bugs in the reassembly function of a number of operating systems, producing buffer overflows that have led, in most cases, to a crash of the attacked system.

4.1.4 Problems that arise from the ambiguity of the reassembly process

Network Intrusion Detection Systems (NIDSs) typically monitor the traffic on a given network with the intent of identifying traffic patterns that might indicate network intrusions.

In the presence of IP fragments, in order to detect illegitimate activity at the transport or application layers (i.e., any protocol layer above the network layer), a NIDS must perform IP fragment reassembly.

In order to correctly assess the traffic, the result of the reassembly function performed by the NIDS should be the same as the reassembly function performed by the intended recipient of the packets.

However a number of factors make the result of the reassembly process ambiguous:

- The IETF specifications are ambiguous as to what should be done when overlapping fragments were received. Thus, in the presence of overlapping data, the system performing the reassembly function is free to either honor the first set of data received, the latest copy received, or any other copy received in between.

- As the specifications do not enforce any specific fragment timeout value, different systems may choose different values for the fragment timeout. This means that given a set of fragments received at some specified time intervals some systems will reassemble the fragments into a full datagram while others may timeout the fragments and therefore drop them.

- As the fragment buffers get full a Denial of Service (DoS) condition will occur unless some action is taken. Many systems flush part of the fragment buffers when some threshold is reached. Thus, depending on fragment load, timing issues and flushing policy, a NIDS may get incorrect assumptions about how (and if) fragments are being reassembled by their intended recipient.

As originally discussed by [Ptacek and Newsham, 1998], these issues can be exploited by attackers to evade intrusion detection systems.

There exist freely available tools to forcefully fragment IP datagrams to help evade Intrusion Detection Systems. Frag router [Song, D., 1999] is an example of such a tool; it allows an attacker to perform all the evasion techniques described in [Ptacek and Newsham, 1998]. Ftester [Barisani, 2006] is a tool that helps to audit systems regarding fragmentation issues.

4.1.5 Problems that arise from the size of the IP fragments

One approach to fragment filtering involves keeping track of the results of applying filter rules to the first fragment (i.e. a **Fragment Offset** of 0) and applying them to subsequent fragments of the same packet. The filtering module would maintain a list of packets indexed by the **Source Address**, **Destination Address**, Protocol and Identification number. When the initial fragment is seen, if the MF bit is set, a list item would be allocated to hold the result of filter access checks. When packets with a non-zero **Fragment Offset** come in, look up the list element with a matching **Source Address/ Destination Address/Protocol/Identification** and apply the stored result (pass or block). When a fragment with a zero **MF** bit is seen, free the list element. Unfortunately, the rules of this type of packet filter can usually be bypassed. [Ziemba et al, 1995] describes the details of the involved technique.

4.1.6 Possible security improvements

Memory allocation for fragment reassembly

A design choice usually has to be made as to how to allocate memory to reassemble the fragments of a given packet. There are basically two options:

- Upon receipt of the first fragment, allocate a buffer large enough to concatenate the payload of each fragment.
- Upon receipt of the first fragment, create the first node of a linked list to which each of the following fragments will be linked. When all fragments have been received, copy the IP payload of each of the fragments (in the correct order) to a separate buffer that will be handed to the protocol being encapsulated in the IP payload.

While the first of the choices might seem to be the most straight-forward, it implies that even when a single small fragment of a given packet is received the amount of memory that will be allocated for that fragment will account for the size of the complete IP datagram, thus using more system resources than what is actually needed.

Furthermore, the only situation in which the actual size of the whole datagram will be known is when the last fragment of the packet is received first, as that is the only packet from which the total size of the IP datagram can be asserted. Otherwise memory should be allocated for largest possible packet size (65535 bytes).

The IP module should also enforce a limit on the amount of memory that can be allocated for IP fragments, as well as a limit on the number of fragments that will be allowed in the system at any time. This will basically limit the resources spent on the reassembly process and prevent an attacker from trashing the whole system memory.

Furthermore, the IP module should keep a different buffer for IP fragments than for complete IP datagrams. This will basically separate the effects of fragment attacks on non-fragmented traffic. Most TCP/IP implementations, such as that in Linux and those in BSD-derived systems, already implement this.

[Jones, 2002] contains an analysis about the amount of memory that may be needed for the fragment reassembly buffer depending on a number of network characteristics.

Flushing the fragment buffer

In the case of those attacks that aim to consume the memory buffers used for fragments, and those that aim to cause a collision of IP **Identification** numbers, there are a number of counter-measures that can be implemented.

The IP module should enforce a limit on the amount of memory that can be allocated for IP fragments, as well as a limit on the number of fragments that will be allowed in the system at any time. This will basically limit the resources spent on the reassembly process, and prevent an attacker from trashing the whole system memory.

Additionally, the IP module should keep a different buffer for IP fragments than for complete IP datagrams. This will basically separate the effects of fragment attacks on non-fragmented traffic. Most TCP/IP implementations, such as that in Linux and those in BSD-derived systems, already implement this.

Even with these counter-measures in place there is still the issue of what to do when the buffer used for IP fragments get full. Basically, if the fragment buffer is full, no instance of communication that relies on fragmentation will be able to progress.

Unfortunately there are not many options for reacting to this situation. If nothing is done all the instances of communication that rely on fragmentation will experience a denial of service. Thus, the only thing that can be done is flush all or part of the fragment buffer on the premise that legitimate traffic will be able to make use of the freed buffer space to allow communication flows to progress.

There are a number of factors that should be taken into consideration when flushing the fragment buffer. First, if a fragment of a given packet (i.e., fragment with a given **Identification** number) is flushed, all the other fragments that correspond to the same datagram should be flushed. As in order for a packet to be reassembled all of its fragments must be received by the system performing the reassembly function. Flushing only a subset of the fragments of a given packet would keep the corresponding buffers tied to fragments that would never reassemble into a complete datagram. Care must be taken so that, in the event that subsequent buffer flushes need to be performed, it is not always the same set of fragments that get dropped as such a behavior would probably cause a selective Denial of Service (DoS) to the traffic flows to which that set of fragments belong.

Many TCP/IP implementations define a threshold for the number of fragments that, when reached, trigger a fragment-buffer flush. Some systems flush 1/2 of the fragment buffer when the threshold is reached. As mentioned before, this is to create some free space in the fragment buffer on the premise that this will allow for new and legitimate fragments to be processed by the IP module, thus letting communication survive the overwhelming situation. On the other hand, the idea of flushing a somewhat large portion of the buffer is to avoid flushing always the same set of packets.

A more selective fragment buffer flushing strategy

One of the difficulties in implementing counter-measures for the fragmentation attacks described in this document is that it is difficult to perform validation checks on the received fragments. For instance, the fragment on which validity checks could be performed, the first fragment, may be not the first fragment to arrive at the destination host.

Fragments can not only arrive out of order because of packet reordering performed by the network, but also because the system (or systems) that fragmented the IP datagram may indeed transmit the fragments out of order. A notable example of this is the Linux TCP/IP stack, which transmits the fragments in reverse order.

> This means that we cannot enforce checks on the fragments for which we allocate reassembly resources, as the first fragment we receive for a given packet may be some other fragment than the first one (the one with an **Fragment Offset** of 0).

However, when we decide to free some space in the fragment buffer, some refinements can be done to the flushing policy. The first thing we would like to do is to stop different types of traffic from interfering with each other. This means, in principle, that we do not want fragmented UDP traffic to interfere with fragmented TCP traffic. In order to implement this traffic separation for the different protocols, a different fragment buffer would be needed, in principle, for each of the 256 different protocols that can be encapsulated in an IP datagram.

We believe a tradeoff is to implement two separate fragment buffers: one for IP datagrams that encapsulate IPsec packets and another for the rest of the traffic. This basically means that traffic not protected by IPsec will not interfere with those flows of communication that are being protected by IPsec.

The processing of each of these two different fragment buffers would be completely independent from each other. In the case of the IPsec fragment buffer, when the buffer needs to be flushed the following refined policy could be applied:

- First, for each packet for which the IPsec header has been received check that the SPI field of the IPsec header corresponds to an existing IPsec Security Association (SA). And probably also check that the IPsec sequence number is valid. If the check fails drop all the fragments that correspond to this packet.

- Second, if the fragment buffer still needs to be flushed drop all the fragments that correspond to packets for which the full IPsec header has not yet been received. The number of packets for which this flushing is performed depends on the amount of free space that needs to be created.

- Third, if there is still not enough space in the fragment buffer after flushing packets with invalid IPsec information (First step), and packets on which validation checks could not be performed (Second step), drop all the fragments that correspond to packets that passed the checks of the first step until the necessary free space is created.

The rationale behind this policy is that, at the point of flushing the fragment buffer, we prefer to keep those packets on which we could successfully perform a number of validation checks over those packets on which those checks failed,or could not be performed.

By checking both the IPsec SPI and the IPsec sequence number, it is virtually impossible for an attacker that is off-path to perform a Denial of Service attack to communication flows being protected by IPsec.

Unfortunately some TCP/IP stacks, when performing fragmentation, send the corresponding fragments in reverse order. In such cases, at the point of flushing the fragment buffer, legitimate fragments will receive the same treatment as the possible forged fragments.

This refined flushing policy provides an increased level of protection against this type of resource exhaustion attack, while not making the situation of out-of-order IPsec-secured traffic worse than with the simplified flushing policy described in the previous section.

Reducing the fragment timeout

RFC 1122 [Braden, 1989] states that the reassembly timeout should be a fixed value between 60 and 120 seconds. The rationale behind these long timeout values is that they should accommodate any path characteristics such as long-delay paths. However it must be noted that this timer is really measuring inter-fragment delays, or more specifically, fragment jitter.

If all fragments take paths of similar characteristics, the inter-fragment delay will usually be a few seconds at most. Nevertheless, even if fragments take different paths of different characteristics the recommended 60 to 120 seconds is excessive.

Some systems have already reduced the fragment timeout to 30 seconds [Linux, 2006]. The fragment timeout could probably be further reduced to approximately 15 seconds although further research on this issue is necessary.

It should be noted that in network scenarios of long-delay and high-bandwidth (usually referred to as “Long-Fat Networks”), using a long fragment timeout would likely increase the probability of collision of IP ID numbers. In such scenarios it is mandatory to avoid the use of fragmentation with techniques such as PMTUD [Mogul and Deering, 1990] or PLPMTUD [Mathis and Heffner, 2007].

Counter-measure for some IDS evasion techniques

[Shankar and Paxson, 2003] introduces a technique named “Active Mapping” that prevents evasion of a NIDS by acquiring sufficient knowledge about the network being monitored to assess which packets will arrive at the intended recipient and how they will be interpreted by it. [Novak, 2005] describes some techniques that are applied by the Snort NIDS to avoid evasion.

Counter-measure for firewall-rules bypassing

One of the classical techniques to bypass firewall rules involves sending packets in which the header of the encapsulated protocol is fragmented. Even when it would be legal (as far as the IETF specifications are concerned) to receive such packets, the MTUs of the network technologies used in practice are not so small as to require the header of the encapsulated protocol to be fragmented. Therefore, the system performing reassembly should drop all packets which fragment the upper-layer protocol header. The necessary information to perform this check could be stored by the IP module together with the rest of the upper-layer protocol information.

Additionally, given that many middle-boxes such as firewalls create state according to the contents of the first fragment of a given packet, it is best that in the event an end-system receives overlapping fragments it honors the information contained in the fragment that was received first.

RFC 1858 [Ziemba et al, 1995] describes the abuse of IP fragmentation to bypass firewall rules. RFC 3128 [Miller, 2001] corrects some errors in RFC 1858.

4.2 Forwarding

4.2.1 Precedence-ordered queue service

Section 5.3.3.1 of RFC 1812 [Baker, 1995] states that routers should implement precedence-ordered queue service. This means that when a packet is selected for output on a (logical) link, the packet of highest precedence that as been queued for that link is sent. Section 5.3.3.2 of RFC 1812 advices routers to default to maintaining strict precedence-ordered service.

Given that it is trivial to forge the IP precedence field of the IP header, an attacker could simply forge a high precedence number in the packets it sends to illegitimately get better network service. If precedence-ordered queued service is not required in a particular network infrastructure it should be disabled and all packets would receive the same type of service, despite the values in their **Type of Service** or **Differentiated Services** fields.

When Precedence-ordered queue service is required in the network infrastructure in order to mitigate the attack vector discussed in the previous paragraph, edge routers or switches should be configured to police and remark the **Type of Service** or **Differentiated Services** values according to the type of service at which each end-system has been allowed to send packets.

Bullet 4 of Section 5.3.3.3 of RFC 1812 states that routers "MUST NOT change precedence settings on packets it did not originate". However, given the security implications of the Precedence field it is fair for routers, switches or other middle-boxes, particularly those in the network edge, to overwrite the **Type of Service** (or **Differentiated Services**) field of the packets they are forwarding according to a configured network policy.

Section 5.3.3.1 and Section 5.3.6 of RFC 1812 states that if precedence-ordered queue service is implemented and enabled the router "MUST NOT discard a packet whose precedence is higher than that of a packet that is not discarded". While this recommendation makes sense given the semantics of the Precedence field, it is important to note that it would be simple for an attacker to send packets with forged high Precedence value to congest some internet router(s) and cause all (or most) traffic with a lower Precedence value to be discarded.

4.2.2 Weak Type of Service

Section 5.2.4.3 of RFC 1812 describes the algorithm for determining the next-hop address (i.e., the forwarding algorithm). Bullet 3, "Weak TOS", addresses the case in which routes contain a "type of service" attribute. It states that in case a packet contains a non-default TOS (i.e., 0000) only routes with the same **TOS** or with the default TOS should be considered for forwarding that packet. However, this means that amongst the longest match routes for a packet are routes with a TOS other than the one contained in the received packet and no routes with the default TOS, thus the corresponding packet would be dropped. This may or may not be a desired behavior.

An alternative to this would be, in cases amongst the "longest match" routes there are only routes with non-default type of services which do not match the **TOS** contained in the received packet,, to use a route with any other **TOS**. While this route would most likely not be able to address the type of service requested by packet, it would, at least, provide a "best effort" service.

It must be noted that Section 5.3.2 of RFC 1812 allows for routers for not honoring the **TOS** field. Therefore, the proposed alternative behavior is still compliant with the IETF specifications.

While officially specified in the RFC series, TOS-based routing is not widely deployed in the Internet.

4.2.3 Address Resolution

In the case of broadcast link-layer technologies, in order for a system to transfer an IP datagram it must usually first map an IP address to the corresponding link-layer address (for example, by means of the ARP protocol [Plummer, 1982]). This means that while this operation is being performed the packets that would require such a mapping would need to be kept in memory. This may happen both in the case of hosts and in the case of routers.

This situation might be exploited by an attacker which could send a large amount of packets to a non-existent host which would supposedly be directly connected to the attacked router. While trying to map the corresponding IP address into a link-layer address, the attacked router would keep in memory all the packets that would need to make use of that link-layer address. At the point in which the mapping function times out, depending on the policy implemented by the attacked router, only the packet that triggered the call to the mapping function might be dropped. In that case, the same operation would be repeated for every packet destined to the non-existent host. Depending on the timeout value for the mapping function this situation might lead to the router buffers to run out of free space with the consequence that incoming legitimate packets would have to be dropped, or that legitimate packets already stored in the router's buffers might get dropped. Both of these situations would lead either to a complete Denial of Service or to a degradation of the network service.

One counter-measure to this problem would be to drop at the point the mapping function times out all the packets destined to the address that timed out. In addition, a "negative cache entry" might be kept in the module performing the matching function, so that for some amount of time the mapping function would return an error when the IP module requests to perform a mapping for some address for which the mapping has recently timed out.

A common implementation strategy for routers is that when a packet is received that requires an ARP request to be performed before the packet can be forwarded, the packet is dropped and the router is then engaged in the ARP procedure.

4.2.4 Dropping packets

In some scenarios it may be necessary for a host or router to drop packets from the output queue. In the event one of such packets happens to be an IP fragment, and there were other fragments of the same packet in the queue, those other fragments should also be dropped. The rationale for this policy is that it is nonsensical to spend system resources on those other fragments because as long as one fragment is missing it will be impossible for the receiving system to reassemble them into a complete IP datagram.

Some systems have been known to drop just a subset of fragments of a given datagram, leading to a denial of service condition, as only a subset of all the fragments of the packets were actually transferred to the next hop.

4.3 Addressing

4.3.1 Unreachable addresses

It is important to understand that while there are some addresses that are supposed to be unreachable from the public Internet (such as those described in RFC 1918 [Rekhter et al, 1996], or the "loopback" address), there are a number of tricks an attacker can perform to reach them (e.g. exploit the LSRR or SSRR IP options). Therefore, when applicable, packet filtering should be performed at organisational network boundary to ensure that those addresses will be unreachable.

4.3.2 Private address space

The Internet Assigned Numbers Authority (IANA) has reserved the following three blocks of the IP address space for private internets:

- 10.0.0.0 - 10.255.255.255 (10/8 prefix)
- 172.16.0.0 - 172.31.255.255 (172.16/12 prefix)
- 192.168.0.0 - 192.168.255.255 (192.168/16 prefix)

Use of these address blocks is described in RFC 1918 [Rekhter et al, 1996].

Where applicable, packet filtering should be performed at the organisational perimeter to assure that these addresses are not reachable from outside the enterprise network.

4.3.3 Class D addresses (224/4 address block)

The Class D addresses correspond to the 224/4 address block and are used for Internet multicast. Therefore if a packet is received with a Class D address as the **Source Address** it should be dropped. Additionally, if an IP packet with a multicast **Destination Address** is received for a connection-oriented protocol (e.g. TCP) the packet should be dropped.

4.3.4 Class E addresses (240/4 address block)

The Class E addresses correspond to the 240/4 address block and are currently reserved for experimental use. As a result, a number of implementations discard packets that contain a Class E address as the **Source Address** or **Destination Address**.

There is ongoing work to reclassify the Class E (240/4) address block as usable unicast address spaces [Fuller et al, 2008a] [Fuller et al, 2008b] [Wilson et al, 2007]. Therefore, we recommend implementations to accept addresses in the 240/4 block as valid addresses for the **Source Address** and **Destination Address**.

It should be noted that the broadcast address 255.255.255.255 still must be treated as indicated in Section 4.3.7 of this document.

4.3.5 Broadcast and multicast addresses, and connection-oriented protocols

For connection-oriented protocols, such as TCP, shared state is maintained between only two endpoints at a time. Therefore, if an IP packet with a multicast (or broadcast) **Destination Address** is received for a connection-oriented protocol (e.g. TCP), the packet should be dropped.

4.3.6 Broadcast and network addresses

The IETF specifications did not originally permit IP addresses to have the value 0 or -1 for any of the Host number, network number or subnet number fields, except for the cases indicated in Section 4.3.7. However this changed fundamentally with the deployment of Classless Inter-Domain Routing (CIDR) [Fuller and Li, 2006] because with CIDR a system cannot know what the subnet mask is for a particular IP address.

Many systems now allow administrators to use the values 0 or -1 for those fields. Despite that according to the IETF specifications these addresses are illegal, modern IP implementations should consider these addresses to be valid.

4.3.7 Special Internet addresses

RFC 1812 [Baker, 1995] discusses the use of some special internet addresses, which is of interest to perform some sanity checks on the **Source Address** and **Destination Address** fields of an IP packet. It uses the following notation for an IP address:

{ <Network-prefix>, <Host-number> }

RFC 1122 [Braden, 1989] contained a similar discussion of special internet addresses, including some of the form { <Network-prefix>, <Subnet-number>, <Host-number> }. However, as explained in Section 4.2.2.11 of RFC 1812, in a CIDR world, the subnet number is clearly an extension of the network prefix and cannot be interpreted without the remainder of the prefix.

{0, 0}

This address means “this host on this network”. It is meant to be used only during the initialisation procedure, by which the host learns its own IP address. If a packet is received with 0.0.0.0 as the **Source Address** for any purpose other than bootstrapping, the corresponding packet should be silently dropped. If a packet is received with 0.0.0.0 as the **Destination Address** it should be silently dropped.

{0, Host number}

This address means “the specified host, in this network”. As in the previous case, it is meant to be used only during the initialisation procedure by which the host learns its own IP address. If a packet is received with such an address as the **Source Address** for any purpose other than bootstrapping, it should be dropped. If a packet is received with such an address as the **Destination Address** it should be dropped.

{-1, -1}

This address is the local broadcast address. It should not be used as a source IP address. If a packet is received with 255.255.255.255 as the **Source Address** it should be dropped.

Some systems, when receiving an ICMP echo request, for example, will use the **Destination Address** in the ICMP echo request packet as the **Source Address** of the response they send (in this case, an ICMP echo reply). Thus, when such systems receive a request sent to a broadcast address, the **Source Address** of the response will contain a broadcast address. This should be considered a bug, rather than a malicious use of the limited broadcast address.

{Network number, -1}

This is the directed broadcast to the specified network. As recommended by RFC 2644 [Senie, 1999], routers should not forward network-directed broadcasts. This avoids the corresponding network from being utilised as, for example, a “smurf amplifier” [CERT, 1998a].

As noted in Section 4.3.6 of this document, many systems now allow administrators to configure these addresses as unicast addresses for network interfaces. In such scenarios routers should forward these addresses as if they were traditional unicast addresses.

In some scenarios a host may have knowledge about a particular IP address being a network-directed broadcast address rather than a unicast address (e.g. that IP address is configured on the local system as a “broadcast address”). If a system can infer the **Source Address** of a received packet is a network-directed broadcast address the packet should be dropped.

As noted in Section 4.3.6 of this document, with the deployment of CIDR [Fuler, 2006], it may be difficult for a system to infer whether a particular IP address is a broadcast address.

{127, any}

This is the internal host loopback address. Any packet that arrives on any physical interface containing this address as the **Source Address**, the **Destination Address**, or as part of a source route (either LSRR or SSRR) should be dropped.

For example, packets with a **Destination Address** in the 127.0.0.0/8 address block that are received on an interface other than loopback should be silently dropped. Packets received on any interface other than loopback with a **Source Address** corresponding to the system receiving the packet should also be dropped.

5. References

Anderson, J. 2001. *An Analysis of Fragmentation Attacks*. Available at:
http://www.ouah.org/fragma.html

Almquist, P. 1992. *Type of Service in the Internet Protocol Suite.* RFC 1349.

Baker, F. 1995. *Requirements for IP Version 4 Routers.* RFC 1812.

Baker, F., Savola, P. 2004. *Ingress Filtering for Multihomed Networks.* RFC 3704.

Barisani, A. 2006. *FTester - Firewall and IDS testing tool.* Available at:
http://dev.inversepath.com/trac/ftester

Bellovin, S. M. 1989. *Security Problems in the TCP/IP Protocol Suite.* Computer Communication Review, Vol. 19, No. 2, pp. 32-48.

Bellovin, S. M. 2002. *A Technique for Counting NATted Hosts.* IMW'02, Nov. 6-8, 2002, Marseille, France.

Bendi, 1998. Boink exploit. Available at:
http://www.insecure.org/sploits/95.NT.fragmentation.bonk.html

Biondi, P., Ebalard, A. 2007. *IPv6 Routing Header Security.* CanSecWest 2007 Security Conference. Available at: http://www.secdev.org/conf/IPv6_RH_security-csw07.pdf

Blake, S., Black, D., Carlson, M., Davies, E., Wang, Z., and Weiss, W., 1998. *An Architecture for Differentiated Services.* RFC 2475.

Braden, R. 1989. *Requirements for Internet Hosts -- Communication Layers.* RFC 1122.

Cerf, V., and Kahn, R. 1974. *A Protocol for Packet Network Intercommunication.* IEEE Transactions on Communications, Vol. 22, No. 5, May 1974, pp. 637-648.

CERT. 1996a. *CERT Advisory CA-1996-01: UDP Port Denial-of-Service Attack.* Available at:
http://www.cert.org/advisories/CA-1996-01.html

CERT. 1996b. *CERT Advisory CA-1996-21: TCP SYN Flooding and IP Spoofing Attacks.* Available at:
http://www.cert.org/advisories/CA-1996-21.html

CERT. 1996c. *CERT Advisory CA-1996-26: Denial-of-Service Attack via ping.* Available at:
http://www.cert.org/advisories/CA-1996-26.html

CERT. 1997. *CERT Advisory CA-1997-28: IP Denial-of-Service Attacks.* Available at:
http://www.cert.org/advisories/CA-1997-28.html

CERT 1998a. *CERT Advisory CA-1998-01: Smurf IP Denial-of-Service Attacks.* Available at:
http://www.cert.org/advisories/CA-1998-01.html

CERT. 1998b. *CERT Advisory CA-1998-13: Vulnerability in Certain TCP/IP Implementations.* Available at: http://www.cert.org/advisories/CA-1998-13.html

CERT. 1999. *CERT Advisory CA-1999-17: Denial-of-Service Tools*. Available at: http://www.cert.org/advisories/CA-1999-17.html

CERT. 2001. *CERT Advisory CA-2001-09: Statistical Weaknesses in TCP/IP Initial Sequence Numbers*. Available at: http://www.cert.org/advisories/CA-2001-09.html

CERT. 2003. *CERT Advisory CA-2003-15 Cisco IOS Interface Blocked by IPv4 Packet.* Available at: http://www.cert.org/advisories/CA-2003-15.html

CIPSO. 1992. *COMMERCIAL IP SECURITY OPTION (CIPSO 2.2).* IETF Internet-Draft (draft-ietf-cipso-ipsecurity-01.txt).

CIPSOWG. 1994. Commercial Internet Protocol Security Option (CIPSO) Working Group. Working Group charter available at: http://www.ietf.org/proceedings/94jul/charters/cipso-charter.html

Cisco. 2003. *Cisco Security Advisory: Cisco IOS Interface Blocked by IPv4 packet*. Available at: http://www.cisco.com/en/US/products/products_security_advisory09186a00801a34c2.shtml

Cisco. 2008. *Cisco IOS Security Configuration Guide, Release 12.2.* Available at: http://www.cisco.com/en/US/docs/ios/12_2/security/configuration/guide/scfipso.html

Clark, D.D. 1982. *IP datagram reassembly algorithms.* RFC 815.

Clark, D.D. 1988. *The Design Philosophy of the DARPA Internet Protocols,* Computer Communication Review, Vol. 18, No.4, pp. 106-114.

daemon9, route, and infinity. 1996. *IP-spoofing Demystified (Trust-Relationship Exploitation)*. Phrack Magazine, Volume Seven, Issue Forty-Eight, File 14 of 18. Available at: http://www.phrack.org/phrack/48/P48-14

Ed3f. 2002. *Firewall spotting and networks analisys with a broken CRC.* Phrack Magazine, Volume 0x0b, Issue 0x3c, Phile #0x0c of 0x10. Available at: http://www.phrack.org/phrack/60/p60-0x0c.txt

Eddy, W. 2007. *TCP SYN Flooding Attacks and Common Mitigations.* RFC 4987.

Ferguson, P., and Senie, D. 2000. *Network Ingress Filtering: Defeating Denial of Service Attacks which employ IP Source Address Spoofing.* RFC 2827.

FIPS. 1994. *Standard Security Label for Information Transfer. Federal Information Processing Standards Publication. FIP PUBS 188.* Available at: http://csrc.nist.gov/publications/fips/fips188/fips188.pdf

Fuller, V., Li, T. 2006. Classless Inter-domain Routing (CIDR): *The Internet Assignment and Aggregation Plan.* RFC 4632.

Fuller, V., Lear, E., Meyer, D. 2008. 240.0.0.0/4: *The Future Begins Now.* Routing SIG Meeting, 25th APNIC Open Policy Meeting, February 25 – 29 2008, Taipei, Taiwan. Slides available at: http://www.apnic.net/meetings/25/program/routing/fuller-240-future.pdf

Fuller, V., Lear, E., Meyer, D. 2008. Reclassifying 240/4 *as usable unicast address space.* IETF Internet-Draft (draft-fuller-240space-00.txt), work in progress.

Fyodor, 2004. *Idle scanning and related IP ID games.* Available at:
http://www.insecure.org/nmap/idlescan.html

GIAC. 2000. *Egress Filtering v 0.2.* Available at:
http://www.sans.org/y2k/egress.htm

Gill, V., Heasley, J., Meyer, D., Savola, P., Pignataro, C. 2007. *The Generalized TTL Security Mechanism (GTSM).* RFC 5082.

Gont, F. 2006. *Advanced ICMP packet filtering.* Available at:
http://www.gont.com.ar/papers/icmp-filtering.html

Gont, F. 2008. *ICMP attacks against TCP*, IETF Internet-Draft (draft-ietf-tcpm-icmp-attacks-03.txt), work in progress.

Graff, C. 1995. *IPv4 Option for Sender Directed Multi-Destination Delivery.* RFC 1770.

Haddad, I., and Zakrzewski, M. 2004. *Security Distribution for Linux Clusters.* Linux Journal. Available at: http://www.linuxjournal.com/article/6943

Heffner, J., Mathis, M., and Chandler, B. 2006. IPv4 *Reassembly Errors at High Data Rates.* RFC 4963.

Humble. 1998. Nestea exploit. Available at:
http://www.insecure.org/sploits/linux.PalmOS.nestea.html

IANA. 2006a. *Ether Types.* Available at:
http://www.iana.org/assignments/ethernet-numbers

IANA. 2006b. *IP Parameters.* Available at:
http://www.iana.org/assignments/ip-parameters

IANA. 2006c. *Protocol Numbers*. Available at:
http://www.iana.org/assignments/protocol-numbers

IRIX. 2008. IRIX 6.5 trusted_networking(7) manual page. Available at:
http://techpubs.sgi.com/library/tpl/cgi-bin/getdoc.cgi?coll=0650&db=man&fname=/usr/share/catman/a_man/cat7/trusted_networking.z

Jones, R. 2002. *A Method Of Selecting Values For the Parameters Controlling IP Fragment Reassembly.* Available at:
ftp://ftp.cup.hp.com/dist/networking/briefs/ip_reass_tuning.txt

Katz, D. 1997. *IP Router Alert Option.* RFC 2113.

Kenney, M. 1996. *The Ping of Death Page.* Available at:
http://www.insecure.org/sploits/ping-o-death.html

Kent, C. and Mogul, J. 1987. *Fragmentation considered harmful*. Proc. SIGCOMM '87, Vol. 17, No. 5, October 1987.

Kent, S. 1991. *U.S. Department of Defense Security Options for the Internet Protocol.* RFC 1108.

Klein, A. 2007. *OpenBSD DNS Cache Poisoning and Multiple O/S Predictable IP ID Vulnerability. Available at:* http://www.trusteer.com/files/OpenBSD_DNS_Cache_Poisoning_and_Multiple_OS_Predictable_IP_ID_Vulnerability.pdf

Kohno, T., Broido, A., and Claffy, K.C. 2005. *Remote Physical Device Fingerprinting.* IEEE Transactions on Dependable and Secure Computing. IEEE Transactions on Dependable and Secure Computing, Vol. 2, No. 2.

LBNL/NRG. 2006. arpwatch tool. Available at:
http://ee.lbl.gov/

Linux. 2006. The Linux Kernel. Available at:
http://www.kernel.org

Malkin, G. 1993. *Traceroute Using an IP Option.* RFC 1393

Mathis, M., and Heffner, J. 2007. *Packetization Layer Path MTU Discovery.* RFC 4821.

Microsoft. 1999. *Microsoft Security Program: Microsoft Security Bulletin (MS99-038). Patch Available for "Spoofed Route Pointer" Vulnerability.* Available at:
http://www.microsoft.com/technet/security/bulletin/ms99-038.mspx

Miller, I. 2001. *Protection Against a Variant of the Tiny Fragment Attack.* RFC 3128.

Mogul, J., Kent, C., Partridge, C., McCloghrie, K. 1988. *IP MTU Discovery Options*. RFC 1063.

Mogul, J., and Deering, S. 1990. *Path MTU Discovery.* RFC 1191.

Nichols, K., Blake, S., Baker, F., and Black, D. 1998. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers.* RFC 2474.

NISCC. 2004. *NISCC Vulnerability Advisory 236929: Vulnerability Issues in TCP.* Available at:
http://www.uniras.gov.uk/niscc/docs/re-20040420-00391.pdf

NISCC. 2005. *NISCC Vulnerability Advisory 532967/NISCC/ICMP: Vulnerability Issues in ICMP packets with TCP payloads.* Available at:
http://www.niscc.gov.uk/niscc/docs/re-20050412-00303.pdf

NISCC. 2006. *NISCC Technical Note 01/2006: Egress and Ingress Filtering.* Available at:
http://www.niscc.gov.uk/niscc/docs/re-20060420-00294.pdf?lang=en

Northcutt, S., and Novak, J. 2000. *Network Intrusion Detection – An Analyst's Handbook. Second Edition.* New Riders Publishing.

Novak, J. 2005. *Target-Based Fragmentation Reassembly*. Available at:
http://www.snort.org/reg/docs/target_based_frag.pdf

OpenBSD. 1998. *OpenBSD Security Advisory: IP Source Routing Problem.* Available at:
http://www.openbsd.org/advisories/sourceroute.txt

Paxson, V., Handley, M., and Kreibich, C. 2001. *Network Intrusion Detection: Evasion, Traffic Normalization, and End-to-End Protocol Semantics.* USENIX Conference, 2001.

Plummer, D.C. 1982. *An Ethernet Address Resolution Protocol.* RFC 826.

Postel, J. 1981. *Internet Protocol. DARPA Internet Program. Protocol Specification.* RFC 791.

Ptacek, T. H., and Newsham, T. N. 1998. *Insertion, Evasion and Denial of Service: Eluding Network Intrusion Detection. Secure Networks, Inc.* Available at:
http://www.aciri.org/vern/Ptacek-Newsham-Evasion-98.ps

Ramakrishnan, K., Floyd, S., and Black, D. 2001. *The Addition of Explicit Congestion Notification (ECN) to IP.* RFC 3168.

Rekhter, Y., Moskowitz, B., Karrenberg, D., de Groot, G. J., and Lear, E. 1996. *Address Allocation for Private Internets.* RFC 1918.

Sanfilippo, S. 1998a. *about the ip header id.* Post to Bugtraq mailing-list, Mon Dec 14 1998. Available at: http://www.kyuzz.org/antirez/papers/ipid.html

Sanfilippo, S. 1998b. *Idle scan*. Post to Bugtraq mailing-list. Available at:
http://www.kyuzz.org/antirez/papers/dumbscan.html

Sanfilippo, S. 1999. *more ip id.* Post to Bugtraq mailing-list. Available at:
http://www.kyuzz.org/antirez/papers/moreipid.html

Savola, P. 2006. *MTU and Fragmentation Issues with In-the-Network Tunneling.* 2006. RFC 4459.

SELinux. 2008. Security Enhanced Linux.
Available at: http://www.nsa.gov/selinux/

Senie, D. 1999. *Changing the Default for Directed Broadcasts in Routers.* RFC 2644.

Shankar, U., and Paxson, V. 2003. *Active Mapping: Resisting NIDS EvasionWithout Altering Traffic.* Available at: http://www.icir.org/vern/papers/activemap-oak03.pdf

Shannon, C., Moore, D., and Claffy, K.C. 2001. *Characteristics of Fragmented IP Traffic on Internet Links.*

Shepler, S., Callaghan, B, Robinson, D., Thurlow, R., Beame, C., Eisler, M., Noveck, D, 2003. *Network File System (NFS) version 4 Protocol.* RFC 3530.

Silbersack, M. J. 2005. *Improving TCP/IP security through randomization without sacrificing interoperability.* EuroBSDCon 2005 Conference. Slides available at: http://www.silby.com/eurobsdcon05/eurobsdcon_slides.pdf

Solaris. 2008. *Solaris Trusted Extensions - Labeled Security for Absolute Protection*. Available at: http://www.sun.com/software/solaris/ds/trusted_extensions.jsp#3

Song, D. 1999. *Frag router tool.* Available at: http://www.anzen.com/research/nidsbench/

St. Johns, M. 1988. *Draft Revised IP Security Option.* RFC 1038.

St. Johns, M., Atkinson, R., and Thomas, G. 2008. *Common Architecture Label IPv6 Security Option (CALIPSO).* IETF Internet-Draft (draft-stjohns-sipso-03.txt), work in progress.

Templin, F. 2006. *Requirements for IP-in-IP Tunnel MTU Assurance*. IETF Internet-Draft (draft-templin-mtuassurance-01.txt), work in progress.

US-CERT. 2001. *US-CERT Vulnerability Note VU#446689: Check Point FireWall-1 allows fragmented packets through firewall if Fast Mode is enabled.* Available at: http://www.kb.cert.org/vuls/id/446689

US-CERT. 2002. *US-CERT Vulnerability Note VU#310387: Cisco IOS discloses fragments of previous packets when Express Forwarding is enabled.* Available at: http://www.kb.cert.org/vuls/id/310387

Watson, Paul. 2004. *Slipping in the Window: TCP Reset Attacks*. CanSecWest 2004 Conference.

Wilson, P., Michaelson, G., Huston, G. 2007. *Redesignation of 240/4 from “Future Use” to “Limited Use for Large Private Internets”*. IETF Internet-Draft (draft-wilson-class-e-01.txt).

Zakrzewski, M., and Haddad, I. 2002. *Linux Distributed Security Module.* Linux Journal. Available at: http://www.linuxjournal.com/article/6215

Ziemba, G., Reed, D., and Traina, P. 1995. *Security Considerations for IP Fragment Filtering.* RFC 1858.