Internet Engineering Task Force (IETF)
Request for Comments: 5927
Category: Informational
ISSN: 2070-1721

F. Gont
UTN/FRH

July 2010

ICMP Attacks against TCP

Abstract

This document discusses the use of the Internet Control Message Protocol (ICMP) to perform a variety of attacks against the Transmission Control Protocol (TCP). Additionally, this document describes a number of widely implemented modifications to TCP's handling of ICMP error messages that help to mitigate these issues.

Status of This Memo

This document is not an Internet Standards Track specification; it is published for informational purposes.

This document is a product of the Internet Engineering Task Force (IETF). It represents the consensus of the IETF community. It has received public review and has been approved for publication by the Internet Engineering Steering Group (IESG). Not all documents approved by the IESG are a candidate for any level of Internet Standard; see Section 2 of RFC 5741.

Information about the current status of this document, any errata, and how to provide feedback on it may be obtained at http://www.rfc-editor.org/info/rfc5927.

Table of Contents

1. Introduction

ICMP [RFC0792] [RFC4443] is a fundamental part of the TCP/IP protocol suite, and is used mainly for reporting network error conditions. However, the current specifications do not recommend any kind of validation checks on the received ICMP error messages, thus allowing a variety of attacks against TCP [RFC0793] by means of ICMP, which include blind connection-reset, blind throughput-reduction, and blind performance-degrading attacks. All of these attacks can be performed even when the attacker is off-path, without the need to sniff the packets that correspond to the attacked TCP connection.

While the possible security implications of ICMP have been known in the research community for a long time, there has never been an official proposal on how to deal with these vulnerabilities. In 2005, a disclosure process was carried out by the UK's National Infrastructure Security Co-ordination Centre (NISCC) (now CPNI, Centre for the Protection of National Infrastructure), with the collaboration of other computer emergency response teams. A large number of implementations were found vulnerable to either all or a subset of the attacks discussed in this document [NISCC][US-CERT]. The affected systems ranged from TCP/IP implementations meant for desktop computers, to TCP/IP implementations meant for core Internet routers.

It is clear that implementations should be more cautious when processing ICMP error messages, to eliminate or mitigate the use of ICMP to perform attacks against TCP [RFC4907].

This document aims to raise awareness of the use of ICMP to perform a variety of attacks against TCP, and discusses several counter-measures that eliminate or minimize the impact of these attacks. Most of the these counter-measures can be implemented while still remaining compliant with the current specifications, as they simply describe reasons for not taking the advice provided in the specifications in terms of "SHOULDs", but still comply with the requirements stated as "MUSTs".

We note that the counter-measures discussed in this document are not part of standard TCP behavior, and this document does not change that state of affairs. The consensus of the TCPM WG (TCP Maintenance and Minor Extensions Working Group) was to document this widespread implementation of nonstandard TCP behavior but to not change the TCP standard.

Section 2 provides background information on ICMP. Section 3 discusses the constraints in the general counter-measures that can be implemented against the attacks described in this document.

Section 4 describes several general validation checks that can be implemented to mitigate any ICMP-based attack. Finally, Section 5, Section 6, and Section 7, discuss a variety of ICMP attacks that can be performed against TCP, and describe attack-specific counter-measures that eliminate or greatly mitigate their impact.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Background

2.1. The Internet Control Message Protocol (ICMP)

The Internet Control Message Protocol (ICMP) is used in the Internet architecture mainly to perform the fault-isolation function, that is, the group of actions that hosts and routers take to determine that there is some network failure [RFC0816].

When an intermediate router detects a network problem while trying to forward an IP packet, it will usually send an ICMP error message to the source system, to inform the source system of the network problem taking place. In the same way, there are a number of scenarios in which an end-system may generate an ICMP error message if it finds a problem while processing a datagram. The received ICMP errors are handed to the corresponding transport-protocol instance, which will usually perform a fault recovery function.

It is important to note that ICMP error messages are transmitted unreliably and may be discarded due to data corruption, network congestion, or rate-limiting. Thus, while they provide useful information, upper-layer protocols cannot depend on ICMP for correct operation.

It should be noted that there are no timeliness requirements for ICMP error messages. ICMP error messages could be delayed for various reasons, and at least in theory could be received with an arbitrarily long delay. For example, there are no existing requirements that a router flush any queued ICMP error messages when it is rebooted.

2.1.1. ICMP for IP version 4 (ICMPv4)

[RFC0792] specifies the Internet Control Message Protocol (ICMP) to be used with the Internet Protocol version 4 (IPv4) -- henceforth "ICMPv4". It defines, among other things, a number of error messages that can be used by end-systems and intermediate systems to report errors to the sending system. The Host Requirements RFC [RFC1122]

classifies ICMPv4 error messages into those that indicate "soft errors", and those that indicate "hard errors", thus roughly defining the semantics of them.

The ICMPv4 specification [RFC0792] also defines the ICMPv4 Source Quench message (type 4, code 0), which is meant to provide a mechanism for flow control and congestion control.

[RFC1191] defines a mechanism called "Path MTU Discovery" (PMTUD), which makes use of ICMPv4 error messages of type 3 (Destination Unreachable), code 4 (fragmentation needed and DF bit set) to allow systems to determine the MTU of an arbitrary internet path.

Finally, [RFC4884] redefines selected ICMPv4 messages to include an extension structure and a length attribute, such that those ICMPv4 messages can carry additional information by encoding that information in the extension structure.

Appendix D of [RFC4301] provides information about which ICMPv4 error messages are produced by hosts, intermediate routers, or both.

2.1.2. ICMP for IP version 6 (ICMPv6)

[RFC4443] specifies the Internet Control Message Protocol (ICMPv6) to be used with the Internet Protocol version 6 (IPv6) [RFC2460].

[RFC4443] defines the "Packet Too Big" (type 2, code 0) error message, which is analogous to the ICMPv4 "fragmentation needed and DF bit set" (type 3, code 4) error message. [RFC1981] defines the Path MTU Discovery mechanism for IP version 6, which makes use of these messages to determine the MTU of an arbitrary internet path.

Finally, [RFC4884] redefines selected ICMPv6 messages to include an extension structure and a length attribute, such that those ICMPv6 messages can carry additional information by encoding that information in the extension structure.

Appendix D of [RFC4301] provides information about which ICMPv6 error messages are produced by hosts, intermediate routers, or both.

2.2. Handling of ICMP Error Messages

The Host Requirements RFC [RFC1122] states in Section 4.2.3.9 that TCP MUST act on an ICMP error message passed up from the IP layer, directing it to the connection that triggered the error.

In order to allow ICMP messages to be demultiplexed by the receiving system, part of the original packet that triggered the message is included in the payload of the ICMP error message. Thus, the receiving system can use that information to match the ICMP error to the transport protocol instance that triggered it.

Neither the Host Requirements RFC [RFC1122] nor the original TCP specification [RFC0793] recommends any validation checks on the received ICMP messages. Thus, as long as the ICMP payload contains the information that identifies an existing communication instance, it will be processed by the corresponding transport-protocol instance, and the corresponding action will be performed.

Therefore, in the case of TCP, an attacker could send a crafted ICMP error message to the attacked system, and, as long as he is able to guess the four-tuple (i.e., Source IP Address, Source TCP port, Destination IP Address, and Destination TCP port) that identifies the communication instance to be attacked, he will be able to use ICMP to perform a variety of attacks.

Generally, the four-tuple required to perform these attacks is not known. However, as discussed in [Watson] and [RFC4953], there are a number of scenarios (notably that of TCP connections established between two BGP routers [RFC4271]) in which an attacker may be able to know or guess the four-tuple that identifies a TCP connection. In such a case, if we assume the attacker knows the two systems involved in the TCP connection to be attacked, both the client-side and the server-side IP addresses could be known or be within a reasonable number of possibilities. Furthermore, as most Internet services use the so-called "well-known" ports, only the client port number might need to be guessed. In such a scenario, an attacker would need to send, in principle, at most 65536 packets to perform any of the attacks described in this document. These issues are exacerbated by the fact that most systems choose the port numbers they use for outgoing connections from a subset of the whole port number space, thus reducing the amount of work needed to successfully perform these attacks.

The need to be more cautious when processing received ICMP error messages in order to mitigate or eliminate the impact of the attacks described in this RFC has been documented by the Internet Architecture Board (IAB) in [RFC4907].

2.3. Handling of ICMP Error Messages in the Context of IPsec

Section 5.2 of [RFC4301] describes the processing of inbound IP traffic in the case of "unprotected-to-protected". In the case of ICMP, when an unprotected ICMP error message is received, it is

matched to the corresponding security association by means of the SPI (Security Parameters Index) included in the payload of the ICMP error message. Then, local policy is applied to determine whether to accept or reject the message and, if accepted, what action to take as a result. For example, if an ICMP Destination Unreachable message is received, the implementation must decide whether to act on it, reject it, or act on it with constraints. Section 8 ("Path MTU/DF Processing") discusses the processing of unauthenticated ICMPv4 "fragmentation needed and DF bit set" (type 3, code 4) and ICMPv6 "Packet Too Big" (type 2, code 0) messages when an IPsec implementation is configured to process (vs. ignore) such messages.

Section 6.1.1 of [RFC4301] notes that processing of unauthenticated ICMP error messages may result in denial or degradation of service, and therefore it would be desirable to ignore such messages. However, it also notes that in many cases, ignoring these ICMP messages can degrade service, e.g., because of a failure to process PMTUD and redirection messages, and therefore there is also a motivation for accepting and acting upon them. It finally states that to accommodate both ends of this spectrum, a compliant IPsec implementation MUST permit a local administrator to configure an IPsec implementation to accept or reject unauthenticated ICMP traffic, and that this control MUST be at the granularity of ICMP type and MAY be at the granularity of ICMP type and code. Additionally, an implementation SHOULD incorporate mechanisms and parameters for dealing with such traffic.

Thus, the policy to apply for the processing of unprotected ICMP error messages is left up to the implementation and administrator.

3. Constraints in the Possible Solutions

If a host wants to perform validation checks on the received ICMP error messages before acting on them, it is limited by the piece of the packet that triggered the error that the sender of the ICMP error message chose to include in the ICMP payload. This constrains the possible validation checks, as the number of bytes of the packet that triggered the error message that is included in the ICMP payload is limited.

For ICMPv4, [RFC0792] states that the IP header plus the first 64 bits of the packet that triggered the ICMPv4 message are to be included in the payload of the ICMPv4 error message. Thus, it is assumed that all data needed to identify a transport protocol instance and process the ICMPv4 error message is contained in the first 64 bits of the transport protocol header. Section 3.2.2 of [RFC1122] states that "the Internet header and at least the first 8 data octets of the datagram that triggered the error" are to be

included in the payload of ICMPv4 error messages, and that "more than 8 octets MAY be sent", thus allowing implementations to include more data from the original packet than those required by the original ICMPv4 specification. The "Requirements for IP Version 4 Routers" RFC [RFC1812] states that ICMPv4 error messages "SHOULD contain as much of the original datagram as possible without the length of the ICMP datagram exceeding 576 bytes".

Thus, for ICMPv4 messages generated by hosts, we can only expect to get the entire IP header of the original packet, plus the first 64 bits of its payload. For TCP, this means that the only fields that will be included in the ICMPv4 payload are the source port number, the destination port number, and the 32-bit TCP sequence number. This clearly imposes a constraint on the possible validation checks that can be performed, as there is not much information available on which to perform them.

This means, for example, that even if TCP were signing its segments by means of the TCP MD5 signature option [RFC2385], this mechanism could not be used as a counter-measure against ICMP-based attacks, because, as ICMP messages include only a piece of the TCP segment that triggered the error, the MD5 [RFC1321] signature could not be recalculated. In the same way, even if the attacked peer were authenticating its packets at the IP layer [RFC4301], because only a part of the original IP packet would be available, the signature used for authentication could not be recalculated, and thus the authentication header in the original packet could not be used as a counter-measure for ICMP-based attacks against TCP.

[RFC4884] updated [RFC0792] and specified that ICMPv4 Destination Unreachable (type 3), Time Exceeded (type 11), and Parameter Problem (type 12) messages that have an ICMP Extension Structure appended include at least 128 octets in the "original datagram" field. This would improve the situation, but at the time of this writing, [RFC4884] is not yet widely deployed for end-systems.

For IPv6, the payload of ICMPv6 error messages includes as many octets from the IPv6 packet that triggered the ICMPv6 error message as will fit without making the resulting ICMPv6 error message exceed the minimum IPv6 MTU (1280 octets) [RFC4443]. Thus, more information is available than in the IPv4 case.

Hosts could require ICMP error messages to be authenticated [RFC4301], in order to act upon them. However, while this requirement could make sense for those ICMP error messages sent by hosts, it would not be feasible for those ICMP error messages generated by routers, as this would imply either that the attacked system should have a security association [RFC4301] with every

existing intermediate system, or that it should be able to establish one dynamically. Current levels of deployment of protocols for dynamic establishment of security associations makes this unfeasible. Additionally, this would require routers to use certificates with paths compatible for all hosts on the network. Finally, there may be some scenarios, such as embedded devices, in which the processing power requirements of authentication might not allow IPsec authentication to be implemented effectively.

4. General Counter-Measures against ICMP Attacks

The following subsections describe a number of mitigation techniques that help to eliminate or mitigate the impact of the attacks discussed in this document. Rather than being alternative counter-measures, they can be implemented together to increase the protection against these attacks.

4.1. TCP Sequence Number Checking

The current specifications do not impose any validity checks on the TCP segment that is contained in the ICMP payload. For instance, no checks are performed to verify that a received ICMP error message has been triggered by a segment that was "in flight" to the destination. Thus, even stale ICMP error messages will be acted upon.

Many TCP implementations have incorporated a validation check such that they react only to those ICMP error messages that appear to relate to segments currently "in flight" to the destination system. These implementations check that the TCP sequence number contained in the payload of the ICMP error message is within the range SND.UNA =< SEG.SEQ < SND.NXT. This means that they require that the sequence number be within the range of the data already sent but not yet acknowledged. If an ICMP error message does not pass this check, it is discarded.

Even if an attacker were able to guess the four-tuple that identifies the TCP connection, this additional check would reduce the possibility of considering a spoofed ICMP packet as valid to Flight_Size/2^^32 (where Flight_Size is the number of data bytes already sent to the remote peer, but not yet acknowledged [RFC5681]). For connections in the SYN-SENT or SYN-RECEIVED states, this would reduce the possibility of considering a spoofed ICMP packet as valid to 1/2^^32. For a TCP endpoint with no data "in flight", this would completely eliminate the possibility of success of these attacks.

This validation check has been implemented in Linux [Linux] for many years, in OpenBSD [OpenBSD] since 2004, and in FreeBSD [FreeBSD] and NetBSD [NetBSD] since 2005.

It is important to note that while this check greatly increases the number of packets required to perform any of the attacks discussed in this document, this may not be enough in those scenarios in which bandwidth is easily available and/or large TCP windows [RFC1323] are in use. Additionally, this validation check does not help to prevent on-path attacks, that is, attacks performed in scenarios in which the attacker can sniff the packets that correspond to the target TCP connection.

It should be noted that, as there are no timeliness requirements for ICMP error messages, the TCP Sequence Number check described in this section might cause legitimate ICMP error messages to be discarded. Also, even if this check is enforced, TCP might end up responding to stale ICMP error messages (e.g., if the Sequence Number for the corresponding direction of the data transfer wraps around).

4.2. Port Randomization

As discussed in the previous sections, in order to perform any of the attacks described in this document, an attacker would need to guess (or know) the four-tuple that identifies the connection to be attacked. Increasing the port number range used for outgoing TCP connections, and randomizing the port number chosen for each outgoing TCP connection, would make it harder for an attacker to perform any of the attacks discussed in this document.

[PORT-RANDOM] recommends that transport protocols randomize the ephemeral ports used by clients, and proposes a number of randomization algorithms.

4.3. Filtering ICMP Error Messages Based on the ICMP Payload

The source address of ICMP error messages does not need to be spoofed to perform the attacks described in this document, as the ICMP error messages might legitimately come from an intermediate system. Therefore, simple filtering based on the source address of ICMP error messages does not serve as a counter-measure against these attacks. However, a more advanced packet filtering can be implemented in middlebox devices such as firewalls and NATs. Middleboxes implementing such advanced filtering look at the payload of the ICMP error messages, and perform ingress and egress packet filtering based on the source address of the IP header contained in the payload of the ICMP error message. As the source address contained in the payload of the ICMP error message does need to be spoofed to perform the attacks described in this document, this kind of advanced filtering serves as a counter-measure against these attacks. As with traditional egress filtering [IP-filtering], egress filtering based on the ICMP payload can help to prevent users of the network being

protected by the firewall from successfully performing ICMP attacks against TCP connections established between external systems. Additionally, ingress filtering based on the ICMP payload can prevent TCP connections established between internal systems from being attacked by external systems. [ICMP-Filtering] provides examples of ICMP filtering based on the ICMP payload.

This filtering technique has been implemented in OpenBSD's Packet Filter [OpenBSD-PF], which has in turn been ported to a number of systems, including FreeBSD [FreeBSD].

5. Blind Connection-Reset Attack

5.1. Description

When TCP is handed an ICMP error message, it will perform its fault recovery function, as follows:

- If the network problem being reported is a "hard error", TCP will abort the corresponding connection.

- If the network problem being reported is a "soft error", TCP will just record this information, and repeatedly retransmit its data until they either get acknowledged, or the connection times out.

The Host Requirements RFC [RFC1122] states (in Section 4.2.3.9) that a host SHOULD abort the corresponding connection when receiving an ICMPv4 error message that indicates a "hard error", and states that ICMPv4 error messages of type 3 (Destination Unreachable), codes 2 (protocol unreachable), 3 (port unreachable), and 4 (fragmentation needed and DF bit set) should be considered as indicating "hard errors". In the case of ICMPv4 port unreachables, the specifications are ambiguous, as Section 4.2.3.9 of [RFC1122] states that TCP SHOULD abort the corresponding connection in response to them, but Section 3.2.2.1 of the same RFC ([RFC1122]) states that TCP MUST abort the connection in response to them.

While [RFC4443] did not exist when [RFC1122] was published, one could extrapolate the concept of "hard errors" to ICMPv6 error messages of type 1 (Destination Unreachable), codes 1 (communication with destination administratively prohibited), and 4 (port unreachable).

Thus, an attacker could use ICMP to perform a blind connection-reset attack by sending any ICMP error message that indicates a "hard error" to either of the two TCP endpoints of the connection. Because of TCP's fault recovery policy, the connection would be immediately aborted.

Some stacks are known to extrapolate ICMP "hard errors" across TCP connections, increasing the impact of this attack, as a single ICMP packet could bring down all the TCP connections between the corresponding peers.

It is important to note that even if TCP itself were protected against the blind connection-reset attack described in [Watson] and [TCPM-TCPSECURE] by means of authentication at the network layer [RFC4301], by means of the TCP MD5 signature option [RFC2385], by means of the TCP-AO [RFC5925], or by means of the mechanism specified in [TCPM-TCPSECURE], the blind connection-reset attack described in this document would still succeed.

5.2. Attack-Specific Counter-Measures

An analysis of the circumstances in which ICMP messages that indicate "hard errors" may be received can shed some light on opportunities to mitigate the impact of ICMP-based blind connection-reset attacks.

ICMPv4 type 3 (Destination Unreachable), code 2 (protocol unreachable)

This ICMP error message indicates that the host sending the ICMP error message received a packet meant for a transport protocol it does not support. For connection-oriented protocols such as TCP, one could expect to receive such an error as the result of a connection-establishment attempt. However, it would be strange to get such an error during the life of a connection, as this would indicate that support for that transport protocol has been removed from the system sending the error message during the life of the corresponding connection.

ICMPv4 type 3 (Destination Unreachable), code 3 (port unreachable)

This error message indicates that the system sending the ICMP error message received a packet meant for a socket (IP address, port number) on which there is no process listening. Those transport protocols that have their own mechanisms for signaling this condition should not be receiving these error messages, as the protocol would signal the port unreachable condition by means of its own mechanisms. Assuming that once a connection is established it is not usual for the transport protocol to change (or be reloaded), it should be unusual to get these error messages.

ICMPv4 type 3 (Destination Unreachable), code 4 (fragmentation needed and DF bit set)

This error message indicates that an intermediate node needed to fragment a datagram, but the DF (Don't Fragment) bit in the IP header was set. It is considered a "soft error" when TCP implements PMTUD, and a "hard error" if TCP does not implement PMTUD. Those TCP/IP stacks that do not implement PMTUD (or have disabled it) but support IP fragmentation/reassembly should not be sending their IP packets with the DF bit set, and thus should not be receiving these ICMP error messages. Some TCP/IP stacks that do not implement PMTUD and that do not support IP fragmentation/reassembly are known to send their packets with the DF bit set, and thus could legitimately receive these ICMP error messages.

ICMPv6 type 1 (Destination Unreachable), code 1 (communication with destination administratively prohibited)

This error message indicates that the destination is unreachable because of an administrative policy. For connection-oriented protocols such as TCP, one could expect to receive such an error as the result of a connection-establishment attempt. Receiving such an error for a connection in any of the synchronized states would mean that the administrative policy changed during the life of the connection. However, in the same way this error condition (which was not present when the connection was established) appeared, it could get solved in the near term.

ICMPv6 type 1 (Destination Unreachable), code 4 (port unreachable)

This error message is analogous to the ICMPv4 type 3 (Destination Unreachable), code 3 (port unreachable) error message discussed above. Therefore, the same considerations apply.

The Host Requirements RFC [RFC1122] states in Section 4.2.3.9 that TCP SHOULD abort the corresponding connection in response to ICMPv4 messages of type 3 (Destination Unreachable), codes 2 (protocol unreachable), 3 (port unreachable), and 4 (fragmentation needed and DF bit set). However, Section 3.2.2.1 states that TCP MUST accept an ICMPv4 port unreachable (type 3, code 3) for the same purpose as a RST. Therefore, for ICMPv4 messages of type 3, codes 2 and 4, there is room to go against the advice provided in the existing specifications, while in the case of ICMPv4 messages of type 3, code 3, there is ambiguity in the specifications that may or may not provide some room to go against that advice.

Based on this analysis, most popular TCP implementations treat all ICMP "hard errors" received for connections in any of the synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK, or TIME-WAIT) as "soft errors". That is, they do not abort the corresponding connection upon receipt of them.

Additionally, they do not extrapolate ICMP errors across TCP connections. This policy is based on the premise that TCP should be as robust as possible. Aborting the connection would be to ignore the valuable feature of the Internet -- that for many internal failures, it reconstructs its function without any disruption of the endpoints [RFC0816].

It should be noted that treating ICMP "hard errors" as "soft errors" for connections in any of the synchronized states may prevent TCP from responding quickly to a legitimate ICMP error message.

It is interesting to note that, as ICMP error messages are transmitted unreliably, transport protocols should not depend on them for correct functioning. In the event one of these messages were legitimate, the corresponding connection would eventually time out. Also, applications may still be notified asynchronously about the error condition, and thus may still abort their connections on their own if they consider it appropriate.

In scenarios such as that in which an intermediate system sets the DF bit in the segments transmitted by a TCP that does not implement PMTUD, or the TCP at one of the endpoints of the connection is dynamically disabled, TCP would only abort the connection after a USER TIMEOUT [RFC0793], losing responsiveness. However, these scenarios are very unlikely in production environments, and it is probably preferable to potentially lose responsiveness for the sake of robustness. It should also be noted that applications may still be notified asynchronously about the error condition, and thus may still abort their connections on their own if they consider it appropriate.

In scenarios of multipath routing or route changes, failures in some (but not all) of the paths may elicit ICMP error messages that would likely not cause a connection abort if any of the counter-measures described in this section were implemented. However, aborting the connection would be to ignore the valuable feature of the Internet -- that for many internal failures, it reconstructs its function without any disruption of the endpoints [RFC0816]. That is, communication should survive if there is still a working path to the destination system [DClark]. Additionally, applications may still be notified asynchronously about the error condition, and thus may still abort their connections on their own if they consider it appropriate.

This counter-measure has been implemented in BSD-derived TCP/IP implementations (e.g., [FreeBSD], [NetBSD], and [OpenBSD]) for more than ten years [Wright][McKusick]. The Linux kernel has also implemented this policy for more than ten years [Linux].

6. Blind Throughput-Reduction Attack

6.1. Description

The Host Requirements RFC [RFC1122] states in Section 4.2.3.9 that hosts MUST react to ICMPv4 Source Quench messages by slowing transmission on the connection. Thus, an attacker could send ICMPv4 Source Quench (type 4, code 0) messages to a TCP endpoint to make it reduce the rate at which it sends data to the other endpoint of the connection. [RFC1122] further adds that the RECOMMENDED procedure is to put the corresponding connection in the slow-start phase of TCP's congestion control algorithm [RFC5681]. In the case of those implementations that use an initial congestion window of one segment, a sustained attack would reduce the throughput of the attacked connection to about SMSS (Sender Maximum Segment Size) [RFC5681] bytes per RTT (round-trip time). The throughput achieved during an attack might be a little higher if a larger initial congestion window is in use [RFC3390].

6.2. Attack-Specific Counter-Measures

As discussed in the "Requirements for IP Version 4 Routers" RFC [RFC1812], research seems to suggest that ICMPv4 Source Quench messages are an ineffective (and unfair) antidote for congestion. [RFC1812] further states that routers SHOULD NOT send ICMPv4 Source Quench messages in response to congestion. Furthermore, TCP implements its own congestion control mechanisms ([RFC5681] [RFC3168]) that do not depend on ICMPv4 Source Quench messages.

Based on this reasoning, a large number of implementations completely ignore ICMPv4 Source Quench messages meant for TCP connections. This behavior has been implemented in, at least, Linux [Linux] since 2004, and in FreeBSD [FreeBSD], NetBSD [NetBSD], and OpenBSD [OpenBSD] since 2005. However, it must be noted that this behavior violates the requirement in [RFC1122] to react to ICMPv4 Source Quench messages by slowing transmission on the connection.

7. Blind Performance-Degrading Attack

7.1. Description

When one IP system has a large amount of data to send to another system, the data will be transmitted as a series of IP datagrams. It is usually preferable that these datagrams be of the largest size that does not require fragmentation anywhere along the path from the source to the destination. This datagram size is referred to as the Path MTU (PMTU) and is equal to the minimum of the MTUs of each hop in the path. A technique called "Path MTU Discovery" (PMTUD) lets IP

systems determine the Path MTU of an arbitrary internet path. [RFC1191] and [RFC1981] specify the PMTUD mechanism for IPv4 and IPv6, respectively.

The PMTUD mechanism for IPv4 uses the Don't Fragment (DF) bit in the IP header to dynamically discover the Path MTU. The basic idea behind the PMTUD mechanism is that a source system assumes that the MTU of the path is that of the first hop, and sends all its datagrams with the DF bit set. If any of the datagrams is too large to be forwarded without fragmentation by some intermediate router, the router will discard the corresponding datagram and will return an ICMPv4 "Destination Unreachable, fragmentation needed and DF set" (type 3, code 4) error message to the sending system. This message will report the MTU of the constricting hop, so that the sending system can reduce the assumed Path-MTU accordingly.

For IPv6, intermediate systems do not fragment packets. Thus, there's an "implicit" DF bit set in every packet sent on a network. If any of the datagrams is too large to be forwarded without fragmentation by some intermediate router, the router will discard the corresponding datagram, and will return an ICMPv6 "Packet Too Big" (type 2, code 0) error message to the sending system. This message will report the MTU of the constricting hop, so that the sending system can reduce the assumed Path-MTU accordingly.

As discussed in both [RFC1191] and [RFC1981], the Path-MTU Discovery mechanism can be used to attack TCP. An attacker could send a crafted ICMPv4 "Destination Unreachable, fragmentation needed and DF set" packet (or their ICMPv6 counterpart) to the sending system, advertising a small Next-Hop MTU. As a result, the attacked system would reduce the size of the packets it sends for the corresponding connection accordingly.

The effect of this attack is two-fold. On one hand, it will increase the headers/data ratio, thus increasing the overhead needed to send data to the remote TCP endpoint. On the other hand, if the attacked system wanted to keep the same throughput it was achieving before being attacked, it would have to increase the packet rate. On virtually all systems, this will lead to an increased processing overhead, thus degrading the overall system performance.

A particular scenario that may take place is one in which an attacker reports a Next-Hop MTU smaller than or equal to the amount of bytes needed for headers (IP header, plus TCP header). For example, if the attacker reports a Next-Hop MTU of 68 bytes, and the amount of bytes used for headers (IP header, plus TCP header) is larger than 68 bytes, the assumed Path-MTU will not even allow the attacked system to send a single byte of application data without

fragmentation. This particular scenario might lead to unpredictable results. Another possible scenario is one in which a TCP connection is being secured by means of IPsec. If the Next-Hop MTU reported by the attacker is smaller than the amount of bytes needed for headers (IP and IPsec, in this case), the assumed Path-MTU will not even allow the attacked system to send a single byte of the TCP header without fragmentation. This is another scenario that may lead to unpredictable results.

For IPv4, the reported Next-Hop MTU could be as small as 68 octets, as [RFC0791] requires every internet module to be able to forward a datagram of 68 octets without further fragmentation. For IPv6, while the required minimum IPv6 MTU is 1280, the reported Next-Hop MTU can be smaller than 1280 octets [RFC2460]. If the reported Next-Hop MTU is smaller than the minimum IPv6 MTU, the receiving host is not required to reduce the Path-MTU to a value smaller than 1280, but is required to include a fragmentation header in the outgoing packets to that destination from that moment on.

7.2. Attack-Specific Counter-Measures

The IETF has standardized a Path-MTU Discovery mechanism called "Packetization Layer Path MTU Discovery" (PLPMTUD) that does not depend on ICMP error messages. Implementation of the aforementioned mechanism in replacement of the traditional PMTUD (specified in [RFC1191] and [RFC1981]) eliminates this vulnerability. However, it can also lead to an increase in PMTUD convergence time.

This section describes a modification to the PMTUD mechanism specified in [RFC1191] and [RFC1981] that has been incorporated in OpenBSD and NetBSD (since 2005) to improve TCP’s resistance to the blind performance-degrading attack described in Section 7.1. The described counter-measure basically disregards ICMP messages when a connection makes progress, without violating any of the requirements stated in [RFC1191] and [RFC1981].

Henceforth, we will refer to both ICMPv4 "fragmentation needed and DF bit set" and ICMPv6 "Packet Too Big" messages as "ICMP Packet Too Big" messages.

In addition to the general validation check described in Section 4.1, these implementations include a modification to TCP’s reaction to ICMP "Packet Too Big" error messages that disregards them when a connection makes progress, and honors them only after the corresponding data have been retransmitted a specified number of times. This means that upon receipt of an ICMP "Packet Too Big"

error message, TCP just records this information, and honors it only when the corresponding data have already been retransmitted a specified number of times.

While this basic policy would greatly mitigate the impact of the attack against the PMTUD mechanism, it would also mean that it might take TCP more time to discover the Path-MTU for a TCP connection. This would be particularly annoying for connections that have just been established, as it might take TCP several transmission attempts (and the corresponding timeouts) before it discovers the PMTU for the corresponding connection. Thus, this policy would increase the time it takes for data to begin to be received at the destination host.

In order to protect TCP from the attack against the PMTUD mechanism, while still allowing TCP to quickly determine the initial Path-MTU for a connection, the aforementioned implementations have divided the traditional PMTUD mechanism into two stages: Initial Path-MTU Discovery and Path-MTU Update.

The Initial Path-MTU Discovery stage is when TCP tries to send segments that are larger than the ones that have so far been sent and acknowledged for this connection. That is, in the Initial Path-MTU Discovery stage, TCP has no record of these large segments getting to the destination host, and thus these implementations believe the network when it reports that these packets are too large to reach the destination host without being fragmented.

The Path-MTU Update stage is when TCP tries to send segments that are equal to or smaller than the ones that have already been sent and acknowledged for this connection. During the Path-MTU Update stage, TCP already has knowledge of the estimated Path-MTU for the given connection. Thus, in this case, these implementations are more cautious with the errors being reported by the network.

In order to allow TCP to distinguish segments between those performing Initial Path-MTU Discovery and those performing Path-MTU Update, two new variables are introduced to TCP: maxsizesent and maxsizeacked.

The maxsizesent variable holds the size (in octets) of the largest packet that has so far been sent for this connection. It is initialized to 68 (the minimum IPv4 MTU) when the underlying Internet Protocol is IPv4, and is initialized to 1280 (the minimum IPv6 MTU) when the underlying Internet Protocol is IPv6. Whenever a packet larger than maxsizesent octets is sent, maxsizesent is set to that value.

On the other hand, maxsizeacked holds the size (in octets) of the largest packet (data, plus headers) that has so far been acknowledged for this connection. It is initialized to 68 (the minimum IPv4 MTU) when the underlying Internet Protocol is IPv4, and is initialized to 1280 (the minimum IPv6 MTU) when the underlying Internet Protocol is IPv6. Whenever an acknowledgement for a packet larger than maxsizeacked octets is received, maxsizeacked is set to the size of that acknowledged packet. Note that because of TCP's cumulative acknowledgement, a single ACK may acknowledge the receipt of more than one packet. When that happens, the algorithm may "incorrectly" assume it is in the "Path-MTU Update" stage, rather than the "Initial Path-MTU Discovery" stage (as described below).

Upon receipt of an ICMP "Packet Too Big" error message, the Next-Hop MTU claimed by the ICMP message (henceforth "claimedmtu") is compared with maxsizesent. If claimedmtu is larger than maxsizesent, then the ICMP error message is silently discarded. The rationale for this is that the ICMP error message cannot be legitimate if it claims to have been triggered by a packet larger than the largest packet we have so far sent for this connection.

If this check is passed, claimedmtu is compared with maxsizeacked. If claimedmtu is equal to or larger than maxsizeacked, TCP is supposed to be at the Initial Path-MTU Discovery stage, and thus the ICMP "Packet Too Big" error message is honored immediately. That is, the assumed Path-MTU is updated according to the Next-Hop MTU claimed in the ICMP error message. Also, maxsizesent is reset to the minimum MTU of the Internet Protocol in use (68 for IPv4, and 1280 for IPv6).

On the other hand, if claimedmtu is smaller than maxsizeacked, TCP is supposed to be in the Path-MTU Update stage. At this stage, these implementations are more cautious with the errors being reported by the network, and therefore just record the received error message, and delay the update of the assumed Path-MTU.

To perform this delay, one new variable and one new parameter are introduced to TCP: nsegrto and MAXSEGRTO. The nsegrto variable holds the number of times a specified segment has timed out. It is initialized to zero, and is incremented by one every time the corresponding segment times out. MAXSEGRTO specifies the number of times a given segment must time out before an ICMP "Packet Too Big" error message can be honored, and can be set, in principle, to any value greater than or equal to 0.

Thus, if nsegrto is greater than or equal to MAXSEGRTO, and there's a pending ICMP "Packet Too Big" error message, the corresponding error message is processed. At that point, maxsizeacked is set to claimedmtu, and maxsizesent is set to 68 (for IPv4) or 1280 (for IPv6).

If, while there is a pending ICMP "Packet Too Big" error message, the TCP SEQ claimed by the pending message is acknowledged (i.e., an ACK that acknowledges that sequence number is received), then the "pending error" condition is cleared.

The rationale behind performing this delayed processing of ICMP "Packet Too Big" messages is that if there is progress on the connection, the ICMP "Packet Too Big" errors must be a false claim. By checking for progress on the connection, rather than just for staleness of the received ICMP messages, TCP is protected from attack even if the offending ICMP messages are "in window", and as a corollary, is made more robust to spurious ICMP messages triggered by, for example, corrupted TCP segments.

MAXSEGRTO can be set, in principle, to any value greater than or equal to 0. Setting MAXSEGRTO to 0 would make TCP perform the traditional PMTUD mechanism defined in [RFC1191] and [RFC1981]. A MAXSEGRTO of 1 provides enough protection for most cases. In any case, implementations are free to choose higher values for this constant. MAXSEGRTO could be a function of the Next-Hop MTU claimed in the received ICMP "Packet Too Big" message. That is, higher values for MAXSEGRTO could be imposed when the received ICMP "Packet Too Big" message claims a Next-Hop MTU that is smaller than some specified value. Both OpenBSD and NetBSD set MAXSEGRTO to 1.

In the event a higher level of protection is desired at the expense of a higher delay in the discovery of the Path-MTU, an implementation could consider TCP to always be in the Path-MTU Update stage, thus always delaying the update of the assumed Path-MTU.

Section 7.3 shows this counter-measure in action. Section 7.4 shows this counter-measure in pseudo-code.

It is important to note that the mechanism described in this section is an improvement to the current Path-MTU discovery mechanism, to mitigate its security implications. The current PMTUD mechanism, as specified by [RFC1191] and [RFC1981], still suffers from some functionality problems [RFC2923] that this document does not aim to address. A mechanism that addresses those issues is described in [RFC4821].

7.3. The Counter-Measure for the PMTUD Attack in Action

This section illustrates the operation of the counter-measure for the ICMP attack against the PMTUD mechanism that has been implemented in OpenBSD and NetBSD. It shows both how the fix protects TCP from being attacked and how the counter-measure works in normal scenarios. As discussed in Section 7.2, this section assumes the PMTUD-specific counter-measure is implemented in addition to the TCP sequence number checking described in Section 4.1.

Figure 1 illustrates a hypothetical scenario in which two hosts are connected by means of three intermediate routers. It also shows the MTU of each hypothetical hop. All the following subsections assume the network setup of this figure.

Also, for simplicity's sake, all subsections assume an IP header of 20 octets and a TCP header of 20 octets. Thus, for example, when the PMTU is assumed to be 1500 octets, TCP will send segments that contain, at most, 1460 octets of data.

For simplicity's sake, all the following subsections assume the TCP implementation at Host 1 (H1) has chosen a MAXSEGRTO of 1.

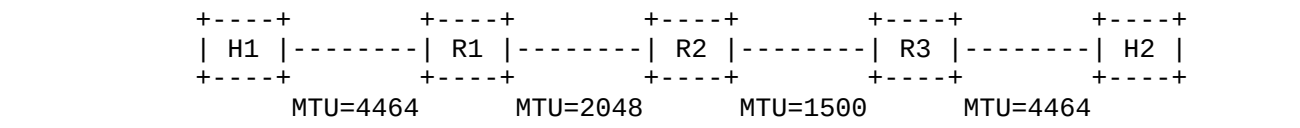

Figure 1: Hypothetical Scenario

7.3.1. Normal Operation for Bulk Transfers

This subsection shows the counter-measure in normal operation, when a TCP connection is used for bulk transfers. That is, it shows how the counter-measure works when there is no attack taking place and a TCP connection is used for transferring large amounts of data. This section assumes that just after the connection is established, one of the TCP endpoints begins to transfer data in packets that are as large as possible.

```
      Host 1                                          Host 2

1.    -->            <SEQ=100><CTL=SYN>             -->
2.    <--      <SEQ=X><ACK=101><CTL=SYN,ACK>        <--
3.    -->       <SEQ=101><ACK=X+1><CTL=ACK>         -->
4.    --> <SEQ=101><ACK=X+1><CTL=ACK><DATA=4424>    -->
5.       <--- ICMP "Packet Too Big" MTU=2048, TCPseq#=101 <--- R1
6.    --> <SEQ=101><ACK=X+1><CTL=ACK><DATA=2008>    -->
7.       <--- ICMP "Packet Too Big" MTU=1500, TCPseq#=101 <--- R2
8.    --> <SEQ=101><ACK=X+1><CTL=ACK><DATA=1460>    -->
9.    <--      <SEQ=X+1><ACK=1561><CTL=ACK>         <--
```

Figure 2: Normal Operation for Bulk Transfers

The nsegrto variable is initialized to zero. Both maxsizeacked and maxsizesent are initialized to the minimum MTU for the Internet Protocol being used (68 for IPv4, and 1280 for IPv6).

In lines 1 to 3, the three-way handshake takes place, and the connection is established. In line 4, H1 tries to send a full-sized TCP segment. As described by [RFC1191] and [RFC1981], in this case, TCP will try to send a segment with 4424 bytes of data, which will result in an IP packet of 4464 octets. Therefore, maxsizesent is set to 4464. When the packet reaches R1, it elicits an ICMP "Packet Too Big" error message.

In line 5, H1 receives the ICMP error message, which reports a Next-Hop MTU of 2048 octets. After performing the TCP sequence number check described in Section 4.1, the Next-Hop MTU reported by the ICMP error message (claimedmtu) is compared with maxsizesent. As it is smaller than maxsizesent, it passes the check, and thus is then compared with maxsizeacked. As claimedmtu is larger than maxsizeacked, TCP assumes that the corresponding TCP segment was performing the Initial PMTU Discovery. Therefore, the TCP at H1 honors the ICMP message by updating the assumed Path-MTU. The maxsizesent variable is reset to the minimum MTU of the Internet Protocol in use (68 for IPv4, and 1280 for IPv6).

In line 6, the TCP at H1 sends a segment with 2008 bytes of data, which results in an IP packet of 2048 octets. The maxsizesent variable is thus set to 2008 bytes. When the packet reaches R2, it elicits an ICMP "Packet Too Big" error message.

In line 7, H1 receives the ICMP error message, which reports a Next-Hop MTU of 1500 octets. After performing the TCP sequence number check, the Next-Hop MTU reported by the ICMP error message (claimedmtu) is compared with maxsizesent. As it is smaller than maxsizesent, it passes the check, and thus is then compared with

maxsizeacked. As claimedmtu is larger than maxsizeacked, TCP assumes that the corresponding TCP segment was performing the Initial PMTU Discovery. Therefore, the TCP at H1 honors the ICMP message by updating the assumed Path-MTU. The maxsizesent variable is reset to the minimum MTU of the Internet Protocol in use.

In line 8, the TCP at H1 sends a segment with 1460 bytes of data, which results in an IP packet of 1500 octets. Thus, maxsizesent is set to 1500. This packet reaches H2, where it elicits an acknowledgement (ACK) segment.

In line 9, H1 finally gets the acknowledgement for the data segment. As the corresponding packet was larger than maxsizeacked, TCP updates maxsizeacked, setting it to 1500. At this point, TCP has discovered the Path-MTU for this TCP connection.

7.3.2. Operation during Path-MTU Changes

Let us suppose a TCP connection between H1 and H2 has already been established, and that the PMTU for the connection has already been discovered to be 1500. At this point, both maxsizesent and maxsizeacked are equal to 1500, and nsegrto is equal to 0. Suppose some time later the PMTU decreases to 1492. For simplicity, let us suppose that the Path-MTU has decreased because the MTU of the link between R2 and R3 has decreased from 1500 to 1492. Figure 3 illustrates how the counter-measure would work in this scenario.

```
    Host 1                                          Host 2

1.                    (Path-MTU decreases)
2.     -->  <SEQ=100><ACK=X><CTL=ACK><DATA=1460>   -->
3.        <--- ICMP "Packet Too Big" MTU=1492, TCPseq#=100 <--- R2
4.                    (Segment times out)
5.     -->  <SEQ=100><ACK=X><CTL=ACK><DATA=1452>   -->
6.     <--         <SEQ=X><ACK=1552><CTL=ACK>       <--
```

Figure 3: Operation during Path-MTU Changes

In line 1, the Path-MTU for this connection decreases from 1500 to 1492. In line 2, the TCP at H1, without being aware of the Path-MTU change, sends a 1500-byte packet to H2. When the packet reaches R2, it elicits an ICMP "Packet Too Big" error message.

In line 3, H1 receives the ICMP error message, which reports a Next-Hop MTU of 1492 octets. After performing the TCP sequence number check, the Next-Hop MTU reported by the ICMP error message (claimedmtu) is compared with maxsizesent. As claimedmtu is smaller than maxsizesent, it is then compared with maxsizeacked. As

claimedmtu is smaller than maxsizeacked (full-sized packets were getting to the remote endpoint), this packet is assumed to be performing Path-MTU Update, and a "pending error" condition is recorded.

In line 4, the segment times out. Thus, nsegrto is incremented by 1. As nsegrto is greater than or equal to MAXSEGRTO, the assumed Path-MTU is updated. The nsegrto variable is reset to 0, maxsizeacked is set to claimedmtu, and maxsizesent is set to the minimum MTU of the Internet Protocol in use.

In line 5, H1 retransmits the data using the updated PMTU, and thus maxsizesent is set to 1492. The resulting packet reaches H2, where it elicits an acknowledgement (ACK) segment.

In line 6, H1 finally gets the acknowledgement for the data segment. At this point, TCP has discovered the new Path-MTU for this TCP connection.

7.3.3. Idle Connection Being Attacked

Let us suppose a TCP connection between H1 and H2 has already been established, and the PMTU for the connection has already been discovered to be 1500. Figure 4 shows a sample time-line diagram that illustrates an idle connection being attacked.

```
    Host 1                                          Host 2

1.    -->   <SEQ=100><ACK=X><CTL=ACK><DATA=50>    -->
2.    <--        <SEQ=X><ACK=150><CTL=ACK>        <--
3.       <--- ICMP "Packet Too Big" MTU=68, TCPseq#=100 <---
4.       <--- ICMP "Packet Too Big" MTU=68, TCPseq#=100 <---
5.       <--- ICMP "Packet Too Big" MTU=68, TCPseq#=100 <---
```

Figure 4: Idle Connection Being Attacked

In line 1, H1 sends its last bunch of data. In line 2, H2 acknowledges the receipt of these data. Then the connection becomes idle. In lines 3, 4, and 5, an attacker sends forged ICMP "Packet Too Big" error messages to H1. Regardless of how many packets it sends and of the TCP sequence number each ICMP packet includes, none of these ICMP error messages will pass the TCP sequence number check described in Section 4.1, as H1 has no unacknowledged data "in flight" to H2. Therefore, the attack does not succeed.

7.3.4. Active Connection Being Attacked after Discovery of the Path-MTU

Let us suppose an attacker attacks a TCP connection for which the PMTU has already been discovered. In this case, as illustrated in Figure 1, the PMTU would be found to be 1500 bytes. Figure 5 shows a possible packet exchange.

```
    Host 1                                          Host 2

1.    -->  <SEQ=100><ACK=X><CTL=ACK><DATA=1460>   -->
2.    -->  <SEQ=1560><ACK=X><CTL=ACK><DATA=1460>  -->
3.    -->  <SEQ=3020><ACK=X><CTL=ACK><DATA=1460>  -->
4.    -->  <SEQ=4480><ACK=X><CTL=ACK><DATA=1460>  -->
5.       <--- ICMP "Packet Too Big" MTU=68, TCPseq#=100 <---
6.    <--       <SEQ=X><CTL=ACK><ACK=1560>        <--
```

Figure 5: Active Connection Being Attacked after Discovery of PMTU

As we assume the PMTU has already been discovered, we also assume both maxsizesent and maxsizeacked are equal to 1500. We assume nsegrto is equal to zero, as there have been no segment timeouts.

In lines 1, 2, 3, and 4, H1 sends four data segments to H2. In line 5, an attacker sends a forged ICMP error message to H1. We assume the attacker is lucky enough to guess both the four-tuple that identifies the connection and a valid TCP sequence number. As the Next-Hop MTU claimed in the ICMP "Packet Too Big" message (claimedmtu) is smaller than maxsizeacked, this packet is assumed to be performing Path-MTU Update. Thus, the error message is recorded.

In line 6, H1 receives an acknowledgement for the segment sent in line 1, before it times out. At this point, the "pending error" condition is cleared, and the recorded ICMP "Packet Too Big" error message is ignored. Therefore, the attack does not succeed.

7.3.5. TCP Peer Attacked when Sending Small Packets Just after the Three-Way Handshake

This section analyzes a scenario in which a TCP peer that is sending small segments just after the connection has been established is attacked. The connection could be in use by protocols such as SMTP [RFC5321] and HTTP [RFC2616], for example, which usually behave like this.

Figure 6 shows a possible packet exchange for such a scenario.

```
      Host 1                                          Host 2

1.    -->           <SEQ=100><CTL=SYN>              -->
2.    <--      <SEQ=X><ACK=101><CTL=SYN,ACK>        <--
3.    -->       <SEQ=101><ACK=X+1><CTL=ACK>         -->
4.    -->  <SEQ=101><ACK=X+1><CTL=ACK><DATA=100>    -->
5.    <--       <SEQ=X+1><ACK=201><CTL=ACK>         <--
6.    -->  <SEQ=201><ACK=X+1><CTL=ACK><DATA=100>    -->
7.    -->  <SEQ=301><ACK=X+1><CTL=ACK><DATA=100>    -->
8.       <--- ICMP "Packet Too Big" MTU=150, TCPseq#=201 <---
```

Figure 6: TCP Peer Attacked when Sending Small Packets Just after the Three-Way Handshake

The nsegrto variable is initialized to zero. Both maxsizesent and maxsizeacked are initialized to the minimum MTU for the Internet Protocol being used (68 for IPv4, and 1280 for IPv6).

In lines 1 to 3, the three-way handshake takes place, and the connection is established. At this point, the assumed Path-MTU for this connection is 4464. In line 4, H1 sends a small segment (which results in a 140-byte packet) to H2. Therefore, maxsizesent is set to 140. In line 5, this segment is acknowledged, and thus maxsizeacked is set to 140.

In lines 6 and 7, H1 sends two small segments to H2. In line 8, while the segments from lines 6 and 7 are still "in flight" to H2, an attacker sends a forged ICMP "Packet Too Big" error message to H1. Assuming the attacker is lucky enough to guess a valid TCP sequence number, this ICMP message will pass the TCP sequence number check. The Next-Hop MTU reported by the ICMP error message (claimedmtu) is then compared with maxsizesent. As claimedmtu is larger than maxsizesent, the ICMP error message is silently discarded. Therefore, the attack does not succeed.

7.4. Pseudo-Code for the Counter-Measure for the Blind Performance-Degrading Attack

This section contains a pseudo-code version of the counter-measure described in Section 7.2 for the blind performance-degrading attack described in Section 7. It is meant as guidance for developers on how to implement this counter-measure.

The pseudo-code makes use of the following variables, constants, and functions:

ack
Variable holding the acknowledgement number contained in the TCP segment that has just been received.

acked_packet_size
Variable holding the packet size (data, plus headers) that the ACK that has just been received is acknowledging.

adjust_mtu()
Function that adjusts the MTU for this connection, according to the ICMP "Packet Too Big" that was last received.

claimedmtu
Variable holding the Next-Hop MTU advertised by the ICMP "Packet Too Big" error message.

claimedtcpseq
Variable holding the TCP sequence number contained in the payload of the ICMP "Packet Too Big" message that has just been received or was last recorded.

current_mtu
Variable holding the assumed Path-MTU for this connection.

drop_message()
Function that performs the necessary actions to drop the ICMP message being processed.

initial_mtu
Variable holding the MTU for new connections, as explained in [RFC1191] and [RFC1981].

maxsizeacked
Variable holding the largest packet size (data, plus headers) that has so far been acked for this connection, as explained in Section 7.2.

maxsizesent
Variable holding the largest packet size (data, plus headers) that has so far been sent for this connection, as explained in Section 7.2.

nsegrto
Variable holding the number of times this segment has timed out, as explained in Section 7.2.

packet_size
Variable holding the size of the IP datagram being sent.

pending_message
Variable (flag) that indicates whether there is a pending ICMP "Packet Too Big" message to be processed.

save_message()
Function that records the ICMP "Packet Too Big" message that has just been received.

MINIMUM_MTU
Constant holding the minimum MTU for the Internet Protocol in use (68 for IPv4, and 1280 for IPv6).

MAXSEGRTO
Constant holding the number of times a given segment must time out before an ICMP "Packet Too Big" error message can be honored.

EVENT: New TCP connection

```
 current_mtu = initial_mtu;
 maxsizesent = MINIMUM_MTU;
 maxsizeacked = MINIMUM_MTU;
 nsegrto = 0;
 pending_message = 0;
```

EVENT: Segment is sent

```
 if (packet_size > maxsizesent)
      maxsizesent = packet_size;
```

EVENT: Segment is received

```
 if (acked_packet_size > maxsizeacked)
      maxsizeacked = acked_packet_size;

 if (pending_message)
      if (ack > claimedtcpseq){
           pending_message = 0;
           nsegrto = 0;
      }
```

EVENT: ICMP "Packet Too Big" message is received

```
 if (claimedmtu <= MINIMUM_MTU)
      drop_message();

 if (claimedtcpseq < SND.UNA || claimedtcpseq >= SND.NXT)
      drop_message();
```

```
    else {
         if (claimedmtu > maxsizesent || claimedmtu >= current_mtu)
              drop_message();

         else {
              if (claimedmtu > maxsizeacked){
                   adjust_mtu();
                   current_mtu = claimedmtu;
                   maxsizesent = MINIMUM_MTU;
              }

              else {
                   pending_message = 1;
                   save_message();
              }
         }
    }
```

EVENT: Segment times out

```
    nsegrto++;

    if (pending_message && nsegrto >= MAXSEGRTO){
         adjust_mtu();
         nsegrto = 0;
         pending_message = 0;
         maxsizeacked = claimedmtu;
         maxsizesent = MINIMUM_MTU;
         current_mtu = claimedmtu;
    }
```

Notes:

All comparisons between sequence numbers must be performed using sequence number arithmetic.

The pseudo-code implements the mechanism described in Section 7.2, the TCP sequence number checking described in Section 4.1, and the validation check on the advertised Next-Hop MTU described in [RFC1191] and [RFC1981].

8. Security Considerations

This document describes the use of ICMP error messages to perform a number of attacks against TCP, and describes a number of widely implemented counter-measures that either eliminate or reduce the impact of these attacks when they are performed by off-path attackers.

Section 4.1 describes a validation check that could be enforced on ICMP error messages, such that TCP reacts only to those ICMP error messages that appear to relate to segments currently "in flight" to the destination system. This requires more effort on the side of an off-path attacker at the expense of possible reduced responsiveness to network errors.

Section 4.2 describes how randomization of TCP ephemeral ports requires more effort on the side of the attacker to successfully exploit any of the attacks described in this document.

Section 4.3 describes how ICMP error messages could possibly be filtered based on their payload, to prevent users of the local network from successfully performing attacks against third-party connections. This is analogous to ingress filtering and egress filtering of IP packets [IP-filtering].

Section 5.2 describes an attack-specific counter-measure for the blind connection-reset attack. It describes the processing of ICMP "hard errors" as "soft errors" when they are received for connections in any of the synchronized states. This counter-measure eliminates the aforementioned vulnerability in synchronized connections at the expense of possible reduced responsiveness in some network scenarios.

Section 6.2 describes an attack-specific counter-measure for the blind throughput-reduction attack. It suggests that the aforementioned vulnerability can be eliminated by ignoring ICMPv4 Source Quench messages meant for TCP connections. This is in accordance with research results that indicate that ICMPv4 Source Quench messages are ineffective and are an unfair antidote for congestion.

Finally, Section 7.2 describes an attack-specific counter-measure for the blind performance-degrading attack. It consists of the validation check described in Section 4.1, with a modification that makes TCP react to ICMP "Packet Too Big" error messages such that they are processed when an outstanding TCP segment times out. This counter-measure parallels the Packetization Layer Path MTU Discovery (PLPMTUD) mechanism [RFC4821]. It should be noted that if this counter-measure is implemented, in some scenarios TCP may respond more slowly to valid ICMP "Packet Too Big" error messages.

A discussion of these and other attack vectors for performing similar attacks against TCP (along with possible counter-measures) can be found in [CPNI-TCP] and [TCP-SECURITY].

9. Acknowledgements

This document was inspired by Mika Liljeberg, while discussing some issues related to [RFC5461] by private e-mail. The author would like to thank (in alphabetical order): Bora Akyol, Mark Allman, Ran Atkinson, James Carlson, Alan Cox, Theo de Raadt, Wesley Eddy, Lars Eggert, Ted Faber, Juan Fraschini, Markus Friedl, Guillermo Gont, John Heffner, Alfred Hoenes, Vivek Kakkar, Michael Kerrisk, Mika Liljeberg, Matt Mathis, David Miller, Toby Moncaster, Miles Nordin, Eloy Paris, Kacheong Poon, Andrew Powell, Pekka Savola, Donald Smith, Pyda Srisuresh, Fred Templin, and Joe Touch for contributing many valuable comments.

Juan Fraschini and the author of this document implemented freely available audit tools to help vendors audit their systems by reproducing the attacks discussed in this document. These tools are available at http://www.gont.com.ar/tools/index.html.

Markus Friedl, Chad Loder, and the author of this document produced and tested in OpenBSD [OpenBSD] the first implementation of the counter-measure described in Section 7.2. This first implementation helped to test the effectiveness of the ideas introduced in this document, and has served as a reference implementation for other operating systems.

The author would like to thank the UK's Centre for the Protection of National Infrastructure (CPNI) -- formerly the National Infrastructure Security Co-ordination Centre (NISCC) -- for coordinating the disclosure of these issues with a large number of vendors and CSIRTs (Computer Security Incident Response Teams).

The author wishes to express deep and heartfelt gratitude to Jorge Oscar Gont and Nelida Garcia, for their precious motivation and guidance.

10. References

10.1. Normative References

[RFC0791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC1122] Braden, R., "Requirements for Internet Hosts - Communication Layers", STD 3, RFC 1122, October 1989.

[RFC1191] Mogul, J. and S. Deering, "Path MTU discovery", RFC 1191, November 1990.

[RFC1812] Baker, F., "Requirements for IP Version 4 Routers", RFC 1812, June 1995.

[RFC1981] McCann, J., Deering, S., and J. Mogul, "Path MTU Discovery for IP version 6", RFC 1981, August 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC4301] Kent, S. and K. Seo, "Security Architecture for the Internet Protocol", RFC 4301, December 2005.

[RFC4443] Conta, A., Deering, S., and M. Gupta, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 4443, March 2006.

[RFC4884] Bonica, R., Gan, D., Tappan, D., and C. Pignataro, "Extended ICMP to Support Multi-Part Messages", RFC 4884, April 2007.

10.2. Informative References

[CPNI-TCP] CPNI, "Security Assessment of the Transmission Control Protocol (TCP)", http://www.cpni.gov.uk/Docs/tn-03-09-security-assessment-TCP.pdf, 2009.

[DClark] Clark, D., "The Design Philosophy of the DARPA Internet Protocols", Computer Communication Review Vol. 18, No. 4, 1988.

[FreeBSD] The FreeBSD Project, http://www.freebsd.org.

[ICMP-Filtering] Gont, F., "Filtering of ICMP error messages", http://www.gont.com.ar/papers/filtering-of-icmp-error-messages.pdf.

[IP-filtering] NISCC, "NISCC Technical Note 01/2006: Egress and Ingress Filtering", http://www.cpni.gov.uk/Docs/re-20060420-00294.pdf, 2006.

[Linux] The Linux Project, "http://www.kernel.org".

[McKusick] McKusick, M., Bostic, K., Karels, M., and J. Quarterman, "The Design and Implementation of the 4.4 BSD Operating System", Addison-Wesley, 1996.

[NISCC] NISCC, "NISCC Vulnerability Advisory 532967/NISCC/ ICMP: Vulnerability Issues in ICMP packets with TCP payloads", http://www.cpni.gov.uk/docs/ re-20050412-00303.pdf?lang=en, 2005.

[NetBSD] The NetBSD Project, "http://www.netbsd.org".

[OpenBSD] The OpenBSD Project, "http://www.openbsd.org".

[OpenBSD-PF] The OpenBSD Packet Filter, "http://www.openbsd.org/faq/pf/".

[PORT-RANDOM] Larsen, M. and F. Gont, "Transport Protocol Port Randomization Recommendations", Work in Progress, April 2010.

[RFC0816] Clark, D., "Fault isolation and recovery", RFC 816, July 1982.

[RFC1321] Rivest, R., "The MD5 Message-Digest Algorithm", RFC 1321, April 1992.

[RFC1323] Jacobson, V., Braden, B., and D. Borman, "TCP Extensions for High Performance", RFC 1323, May 1992.

[RFC2385] Heffernan, A., "Protection of BGP Sessions via the TCP MD5 Signature Option", RFC 2385, August 1998.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[RFC2923] Lahey, K., "TCP Problems with Path MTU Discovery", RFC 2923, September 2000.

[RFC3168] Ramakrishnan, K., Floyd, S., and D. Black, "The Addition of Explicit Congestion Notification (ECN) to IP", RFC 3168, September 2001.

[RFC3390] Allman, M., Floyd, S., and C. Partridge, "Increasing TCP's Initial Window", RFC 3390, October 2002.

[RFC4271] Rekhter, Y., Li, T., and S. Hares, "A Border Gateway Protocol 4 (BGP-4)", RFC 4271, January 2006.

[RFC4821] Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", RFC 4821, March 2007.

[RFC4907] Aboba, B., "Architectural Implications of Link Indications", RFC 4907, June 2007.

[RFC4953] Touch, J., "Defending TCP Against Spoofing Attacks", RFC 4953, July 2007.

[RFC5321] Klensin, J., "Simple Mail Transfer Protocol", RFC 5321, October 2008.

[RFC5461] Gont, F., "TCP's Reaction to Soft Errors", RFC 5461, February 2009.

[RFC5681] Allman, M., Paxson, V., and E. Blanton, "TCP Congestion Control", RFC 5681, September 2009.

[RFC5925] Touch, J., Mankin, A., and R. Bonica, "The TCP Authentication Option", RFC 5925, June 2010.

[TCP-SECURITY] Gont, F., "Security Assessment of the Transmission Control Protocol (TCP)", Work in Progress, February 2010.

[TCPM-TCPSECURE] Ramaiah, A., Stewart, R., and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", Work in Progress, May 2010.

[US-CERT] US-CERT, "US-CERT Vulnerability Note VU#222750: TCP/IP Implementations do not adequately validate ICMP error messages", http://www.kb.cert.org/vuls/id/222750, 2005.

[Watson] Watson, P., "Slipping in the Window: TCP Reset Attacks", CanSecWest Conference, 2004.

[Wright] Wright, G. and W. Stevens, "TCP/IP Illustrated, Volume 2: The Implementation", Addison-Wesley, 1994.

Author's Address

Fernando Gont
Universidad Tecnologica Nacional / Facultad Regional Haedo
Evaristo Carriego 2644
Haedo, Provincia de Buenos Aires 1706
Argentina

Phone: +54 11 4650 8472
EMail: fernando@gont.com.ar
URI: http://www.gont.com.ar