TCP Maintenance and Minor Extensions (tcpm)
Internet-Draft
Intended status: Informational
Expires: April 26, 2007

F. Gont
UTN/FRH

October 23, 2006

ICMP attacks against TCP
draft-ietf-tcpm-icmp-attacks-01.txt

Status of this Memo

Copyright Notice

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) and other similar protocols. It proposes several counter-measures to eliminate or minimize the impact of these attacks.

Table of Contents

1. Introduction

ICMP [RFC0792] 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 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 being 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 vulnerabiliies. Thus, only a few implementations have implemented validation checks on the received ICMP error messages to minimize the impact of these attacks.

Recently, a disclosure process has been carried out by the UK's National Infrastructure Security Co-ordination Centre (NISCC), 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 [I-D.iab-link-indications].

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 suggest reasons for not taking the advice provided in the specifications in terms of "SHOULDs", but still comply with the requirements stated as "MUSTs". Section 5.2, Section 6.2, and Section 7.2 include an explanation of the current requirements and advice relevant to each of the attack-specific counter-measures described in this document.

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 proposes several general validation checks and counter-measures 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 propose 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 raise awareness 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 unreliable, 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.

2.1.1. ICMP for IP version 4 (ICMP)

[RFC0792] specifies the Internet Control Message Protocol (ICMP) to be used with the Internet Protocol version 4 (IPv4). 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 ICMP error messages into those that indicate "soft errors", and those that indicate "hard errors", thus roughly defining the semantics of them.

The ICMP specification [RFC0792] also defines the ICMP 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 ICMP 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.

Appendix D of [RFC4301] provides information about which ICMP 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, that is analogous to the ICMP "fragmentation needed and DF bit set" (type 3, code 4) error message. [RFC1981] defines the Path

MTU Discovery mechanism for IP Version 6, that makes use of these messages to determine the MTU of an arbitrary internet path.

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 a TCP MUST act on an ICMP error message passed up from the IP layer, directing it to the connection that elicited the error.

In order to allow ICMP messages to be demultiplexed by the receiving system, part of the original packet that elicited 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 elicited it.

Neither the Host Requirements RFC [RFC1122] nor the original TCP specification [RFC0793] recommend 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 forged ICMP 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 [Touch-antispoof], there are a number of scenarios (notably that of TCP connections established between two BGP routers), 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. However, as most systems choose the port numbers they use for outgoing connections from a subset of the whole port number space, the amount of packets needed to successfully perform any of the attacks discussed in this document could be further reduced.

It is clear that TCP should be more cautious when processing received ICMP error messages, in order to mitigate or eliminate the impact of the attacks described in this document [I-D.iab-link-indications].

2.3. Handling of ICMP error messages in the context of IPSec

Section 5.2 of [RFC4301] describes the processing 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 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 ICMP "fragmentation needed and DF bit set" (type 3, code 3) and ICMPv6 "Packet Too Big" (type 2, code 0) messages when an IPsec implementation receives 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 PMTU message 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

For ICMPv4, [RFC0792] states that the internet header plus the first 64 bits of the packet that elicited the ICMP message are to be included in the payload of the ICMP error message. Thus, it is assumed that all data needed to identify a transport protocol instance and process the ICMP 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 ICMP 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 ICMP specification. The Requirements for IP Version 4 Routers RFC [RFC1812] states that ICMP error messages "SHOULD contain as much of the original datagram as possible without the length of the ICMP datagram exceeding 576 bytes".

Thus, for ICMP 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 ICMP 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 avalable 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 elicited 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 this mechanism could not be used as a counter-measure aganist ICMP-based attacks against TCP.

For IPv6, the payload of ICMPv6 error messages includes as many octets from the IPv6 packet that elicited the ICMPv6 error message as will fit without making the resulting ICMPv6 packet 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 protocol deployment for dynamic establishment of security associations makes this unfeasible. Also, there may be some cases, such as embedded devices, in which the processing power requirements of authentication could not allow IPSec authentication to be implemented effectively.

Additional considerations for the validation of ICMP error messages can be found in Appendix C

4. General counter-measures against ICMP attacks

There are a number of counter-measures that can be implemented 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. In particular, all TCP implementations should perform the TCP sequence number checking described in Section 4.1.

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 elicited by a segment that was "in flight" to the destination. Thus, even stale ICMP error messages will be acted upon.

TCP should 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 the sequence number should be within the range of the data already sent but not yet acknowledged. If an ICMP error message does not pass this check, it should be 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 [RFC2581]). 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. Therefore, implementation of the attack-specific counter-measures discussed in this document is strongly recommended.

A TCP that implements the TCP sequence number checking as the only validation of ICMP error messages will have the same susceptibility to attacks as the one TCP currently has in the case of TCP-based attacks. Further information on this issue can be found in [Touch-antispoof].

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 conenctions would make it harder for an attacker to perform any of the attacks discussed in this document.

[I-D.larsen-tsvwg-port-randomisation] discusses a number of algorithms to randomize the ephemeral ports used by clients.

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. 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 could be implemented in middlebox devices such as firewalls and NATs as a counter-measure. Middleboxes implementing such advanced filtering would look at the payload of the

ICMP error messages, and would perform ingress and egress packet filtering based on the source IP address of the IP header contained in the payload of the ICMP error message. As the source IP 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 would serve 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 attacks performed by external systems. [ICMP-Filtering] provides examples of ICMP filtering based on the ICMP payload.

This filtering 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:

- o If the network problem being reported is a hard error, TCP will abort the corresponding connection.
- o 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 ICMP error message that indicates a "hard error", and states that ICMP 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 to indicate hard errors. In the case of ICMP 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 [I-D.ietf-tcpm-tcpsecure], by means authentication at the network layer [RFC4301], by means of the TCP MD5 signature option [RFC2385], or by means of the mechanism proposed in [I-D.ietf-tcpm-tcpsecure], the blind connection-reset attack described in this document would still succeed.

5.2. Attack-specific counter-measures

The Host Requirements RFC [RFC1122] states in Section 4.2.3.9 that

TCP SHOULD abort the corresponding connection in response to ICMP messages of type 3, 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 ICMP port unreachable (type 3, code 3) for the same purpose as an RST. Therefore, for ICMP 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 ICMP messages of type 3 code 3 the ambiguity in the specification also allows us to go against the advice provided by the existing specifications, while still remaining compliant with them. Given the hostile environments in which TCP currently operates in, and that advice ICMP provides an attack vector that is easier to exploit than others (such as those discussed in [I-D.ietf-tcpm-tcpsecure]), we believe that the improvements in TCP's resistance to these attacks justify not taking the advice provided by the "SHOULDs" in [RFC1122].

5.2.1. Changing the reaction to hard errors

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

ICMP 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. Thus, it would be fair to treat ICMP protocol unreachable error messages as soft errors if they are meant for connections that are in synchronized states. For TCP, this means TCP would treat ICMP protocol unreachable error messages as soft errors if they are meant for connections that are in any of the synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK or TIME-WAIT).

ICMP 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 which have their own mechanisms for notifying this condition should not be receiving these error messages, as the protocol would signal the port unreachable condition by means of its own messages. Assuming that once a connection is established it is not usual for the transport protocol to change (or be reloaded), it would be fair to treat ICMP port unreachable messages as soft errors when they are meant for a TCP that is in any of the synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK or TIME-WAIT).

ICMP 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 systems that do not implement the PMTUD mechanism should not be sending their IP packets with the DF bit set, and thus should not be receiving these ICMP error messages. Thus, it would be fair for TCPs in any of the synchronized states to treat this ICMP error message as indicating a soft error, therefore not aborting the corresponding connection when such an error message is received. For obvious reasons, those systems implementing the Path-MTU Discovery (PMTUD) mechanism [RFC1191] should not abort the corresponding connection when such an ICMP error message is received.

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, there is no reason to think that in the same way this error condition appeared, it will not get solved in the near term. Therefore, while it would be possible for a firewall to be reconfigured during the life of a connection, it would be fair, for security reasons, to treat these messages as soft errors when they are meant for connections that are in any of the synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK or TIME-WAIT states).

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

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

Therefore, when following the reasoning explained above, TCPs in synchronized states would treat all of the above messages as indicating "soft errors", rather than "hard errors", and thus would not abort the corresponding connection upon receipt of them. This is policy is based on the premise that TCP should be as robust as possible. Reaction to these messages when they are meant for connections in non-synchronized states could still remain as adviced by [RFC1122], as we consider the attack window for connections in the non-synchronized states is very small, and error messages received in these states are more likely indicate that the connection was opened improperly [RFC0816]. Additionally, for the sake of robustness and security, those implementations following the reasoning explained in this section would not extrapolate ICMP errors across TCP connections.

In case the received message were legitimate, it would mean that the error condition appeared during the life of the corresponding connection. However, in many scenarios there is no reason to think that in the same way this error condition appeared, it will not get solved in the near term. Therefore, treating the received ICMP error messages as "soft errors" would make TCP more robust, and could avoid

TCP from aborting a TCP connection unnecesarily. 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 end points [RFC0816].

One scenario in which a host would benefit from treating the so-called ICMP "hard errors" as "soft errors" would be that in which the packets that correspond to a given TCP connection are routed along multiple different paths. Some (but not all) of these paths may be experiencing an error condition, and therefore generating the so-called ICMP hard errors. However, communication should survive if there is still a working path to the destination system [DClark]. Thus, treating the so-called "hard errors" as "soft errors" when a connection is in any of the synchronized states would make TCP achieve this goal.

It is interesting to note that, as ICMP error messages are unreliable, 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 received error messages, 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].

5.2.2. Delaying the connection-reset

An alternative counter-measure would be, in the case of connections in any of the synchronized states, to honor the ICMP error messages only if there is no progress on the connection. Rather than immediately aborting a connection, a TCP would abort a connection only after an ICMP error message indicating a hard error has been received, and the corresponding data have already been retransmitted more than some specified number of times.

The rationale behind this proposed fix is that if a host can make forward progress on a connection, it can completely disregard the "hard errors" being indicated by the received ICMP error messages.

While this counter-measure could be useful, we think that the counter-measure discussed in Section 5.2.1 is easier to implement, and provides increased protection against this type of attack.

5.2.3. Possible drawbacks of the described solutions

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, we consider these senarios very unlikely in production environments, and consider that it is preferebable to potentially lose responsiveness for the sake of robustness. It should also be noted that applications may still be notified asynchronously about the received error messages, 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, as explained above, 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 end points [RFC0816]. Additionally, applications may still be notified asynchronously about the received error messages, and thus may still abort their connections on their own if they consider it appropriate.

6. Blind throughput-reduction attack

6.1. Description

The Host requirements RFC [RFC1122] states that hosts MUST react to ICMP Source Quench messages by slowing transmission on the connection. Thus, an attacker could send ICMP 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 end-point 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 [RFC2581]. 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) [RFC2581] bytes per RTT (round-trip time). The throughput achieved during attack might be a little higher if a larger initial congestion window is in use [RFC3390].

6.2. Attack-specific counter-measures

The Host Requirements RFC [RFC1122] states in Section 4.2.3.9 that hosts MUST react to ICMP Source Quench messages by slowing transmission on the connection. Therefore, the only counter-measures for this attack that can be implemented while still remaining compliant with the existing specifications are the ones discussed in Section 4.

Nevertheless, it must be noted that, as discussed in the Requirements for IP Version 4 Routers RFC [RFC1812], research seems to suggest that ICMP Source Quench is an ineffective (and unfair) antidote for congestion. [RFC1812] further states that routers SHOULD NOT send ICMP Source Quench messages in response to congestion. On the other hand, TCP implements its own congestion control mechanisms [RFC2581] [RFC3168], that do not depend on ICMP Source Quench messages.

Based on this reasoning, a large number of implementations completely ignore ICMP 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, as explained earlier in this section, this behaviour violates the requirement in [RFC1122] to react to ICMP 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

ICMP "Destination Unreachable" (type 3) "fragmentation neeed and DF set" (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 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 forged ICMP "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 end-point. 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 increase in the IRQ (Interrupt ReQuest) rate, thus increasing processor utilization, and degrading the overall system performance.

A particular scenario that may take place is that 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 posible scenario is that 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 low as 68 octets, as

[RFC0791] requires every internet module to be able to forward a datagram of 68 octets without further fragmentation. For IPv6, the reported Next-Hop MTU could be as low as 1280 octets (the minimum IPv6 MTU) [RFC2460].

7.2. Attack-specific counter-measures

This section describes a modification to the PMTUD mechanism specified in [RFC1191] and [RFC1981] that has been implemented in a variety of TCP implementations to improve TCP's resistance to the blind performance-degrading attack described in Section 7.1. The described mechanism basically disregards ICMP messages when a connection makes progress. This modification does not violate any of the requirements stated in [RFC1191] and [RFC1981].

Henceforth, we will refer to both ICMP "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, a counter-measure similar to that described in Section 5.2.2 could be implemented to greatly minimize the impact of this attack.

This would mean that upon receipt of an ICMP "Packet Too Big" error message, TCP would just record this information, and would honor it only when the corresponding data had already been retransmitted a

specified number of times.

While this 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.

We would like to protect TCP from the attack against the PMTUD mechanism, while still allowing TCP to quickly determine the initial Path-MTU for a connection.

To achieve both goals, we can divide 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 it would be fair to 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, it would be fair to be 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 should be introduced to TCP: maxsizeacked and maxsizesent.

maxsizesent would hold the size (in octets) of the largest packet that has so far been sent for this connection. It would be initialized to 68 (the minimum IPv4 MTU) when the underlying internet

protocol is IPv4, and would be initialized to 1280 (the minimum IPv6 MTU) when the underlying internet protocol is IPv6. Whenever a packet larger than maxsizesent octets is sent, maxsizesent should be set to that value.

On the other hand, maxsizeacked would hold the size (in octets) of the largest packet that has so far been acknowledged for this connection. It would be initialized to 68 (the minimum IPv4 MTU) when the underlying internet protocol is IPv4, and would be 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 should be set to the size of that acknowledged packet.

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

If this check is passed, claimedmtu should be 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 should be honored immediately. That is, the assumed Path-MTU should be updated according to the Next-Hop MTU claimed in the ICMP error message.

Also, maxsizesent should be 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, we should be more cautious with the errors being reported by the network, and should 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 should be introduced to TCP: nsegrto and MAXSEGRTO. nsegrto will hold the number of times a specified segment has timed out. It should be

initialized to zero, and should be incremented by one everytime the corresponding segment times out. MAXSEGRRTO should specify the number of times a given segment must timeout 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 correspoing error message should be processed. At that point, maxsizeacked should be set to claimedmtu, and maxsizesent should be 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 should be 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 elicited 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 should provide 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.

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.

Appendix A shows the proposed counter-measure in action. Appendix B shows the proposed counter-measure in pseudo-code.

This behavior has been implemented in NetBSD [NetBSD] and OpenBSD [OpenBSD] since 2005.

It is important to note that the mechanism proposed 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 [I-D.ietf-pmtud-method].

8. Security Considerations

This document describes the use of ICMP error messages to perform a number of attacks against the TCP protocol, and proposes a number of counter-measures that either eliminate or reduce the impact of these attacks.

9. Acknowledgements

This document was inspired by Mika Liljeberg, while discussing some issues related to [I-D.ietf-tcpm-tcp-soft-errors] by private e-mail. The author would like to thank Bora Akyol, Mark Allman, Ran Atkinson, James Carlson, Alan Cox, Theo de Raadt, Ted Faber, Juan Fraschini, Markus Friedl, Guillermo Gont, John Heffner, Vivek Kakkar, Michael Kerrisk, Mika Liljeberg, Matt Mathis, David Miller, Miles Nordin, Eloy Paris, Kacheong Poon, Andrew Powell, Pekka Savola, Pyda Srisuresh, Fred Templin, Joe Touch, and Andres Trapanotto, 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.

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 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.

10.2. Informative References

[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".

[I-D.iab-link-indications]
Aboba, B., "Architectural Implications of Link Indications", draft-iab-link-indications-05 (work in progress), July 2006.

[I-D.ietf-pmtud-method]
Mathis, M. and J. Heffner, "Packetization Layer Path MTU Discovery", draft-ietf-pmtud-method-10 (work in progress), September 2006.

[I-D.ietf-tcpm-tcp-soft-errors]
Gont, F., "TCP's Reaction to Soft Errors", draft-ietf-tcpm-tcp-soft-errors-02 (work in progress), October 2006.

[I-D.ietf-tcpm-tcpsecure]
Stewart, R. and M. Dalal, "Improving TCP's Robustness to Blind In-Window Attacks", draft-ietf-tcpm-tcpsecure-05 (work in progress), June 2006.

[I-D.larsen-tsvwg-port-randomisation]
Larsen, M., "Port Randomisation", draft-larsen-tsvwg-port-randomisation-00 (work in progress), October 2004.

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

[IP-filtering]
NISCC, "NISCC Technical Note 01/2006: Egress and Ingress

Filtering", http://www.niscc.gov.uk/niscc/docs/
re-20060420-00294.pdf?lang=en, 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 4.4BSD Operating System", Addison-Wesley , 1996.

[NISCC] NISCC, "NISCC Vulnerability Advisory 532967/NISCC/ICMP: Vulnerability Issues in ICMP packets with TCP payloads", http://www.niscc.gov.uk/niscc/docs/al-20050412-00308.html?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/".

[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.

[RFC2581] Allman, M., Paxson, V., and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999.

[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.

[RFC2821] Klensin, J., "Simple Mail Transfer Protocol", RFC 2821, April 2001.

[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.

[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", 2004 CanSecWest Conference , 2004.

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

Appendix A. The counter-measure for the PMTUD attack in action

This appendix shows the proposed counter-measure for the ICMP attack against the PMTUD mechanism in action. 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 Appendix assumes the PMTUD-specific counter-measure is implemented in addition to the TCP sequence number checking described in Section 4.1.

Figure 1 illustrates an 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 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 sake, all the following subsections assume the TCP implementation at Host 1 has chosen a a MAXSEGRTO of 1.

```
+-----+        +-----+        +-----+        +-----+        +-----+
| H1  |--------| R1  |--------| R2  |--------| R3  |--------| H2  |
+-----+        +-----+        +-----+        +-----+        +-----+
      MTU=4464       MTU=2048       MTU=1500       MTU=4464
```

Figure 1: Hypothetical scenario

A.1. Normal operation for bulk transfers

This subsection shows the proposed counter-measure in normal operation, when a TCP connection is used for bulk transfers. That is, it shows how the proposed 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

nsegrto 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. maxsizesent 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. maxsizesent 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. maxsizesent 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. maxsizesent is thus 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.

A.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 proposed counter-measure would work in this scenario.

```
     Host 1                                               Host 2

1.                      (Path-MTU decreases)
2.       -->  <SEQ=100><ACK=X><CTL=ACK><DATA=1500>    -->
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 end-point), 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. nsegrto is reset to 0, and 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.

A.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. At 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 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.

A.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 packet 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.

A.5. TCP peer attacked when sending small packets just after the three-way handshake

This section analyzes an scenario in which a TCP peer that is sending small segments just after the connection has been established, is

attacked. The connection could be being used by protocols such as SMTP [RFC2821] and HTTP [RFC2616], for example, which usually behave like this.

Figure 6 shows a possible packet exchange for such 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#=101 <---
```

Figure 6: TCP peer attacked when sending small packets just after the three-way handshake

nsegrto 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. maxsizesent is thus 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.

Appendix B. 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) 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 for been acked far this connection, as explained in Section 7.2

maxsizesent
Variable holding the largest packet size (data, plus headers) that has so for been sent far 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 timeout 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_mesage)
             if (ack > claimedtcpseq){
                  pending_message = 0;
                  nsegrto = 0;
             }
```



```
   EVENT: ICMP "Packet Too Big" message is received

        if (claimedtcpseq < SND.UNA || claimed_TCP_SEQ >= 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 comparisions between sequence numbers must be performed using sequence number arithmethic.
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].

Appendix C. Additional considerations for the validation of ICMP error messages

The checksum of the IP datagram contained in the ICMP payload should be checked to be valid. In case it is invalid, the ICMP error message should be silently dropped.

If a full IP datagram is contained in the ICMP payload, and the IP datagram is authenticated [RFC4301], the signature should be recalculated for that packet. If it doesn't match the one already included in the ICMP payload, the ICMP error message should be silently dropped.

If a full TCP segment is contained in the payload of the ICMP error message, then the first check that should be performed is that the

TCP checksum is valid. Then, if a TCP MD5 option is present, the MD5 signature should be recalculated for the encapsulated packet, and if it doesn't match the one contained in the TCP MD5 option, the ICMP error message should be silently dropped.

Regardless of whether the received ICMP error message contains a full packet or not, if a TCP timestamp option is present, it should be used to validate the error message according to the rules specified in [RFC1323].

It must be noted that most of the checks discussed in this appendix imply that the ICMP error message contains more data than just the full IP header and the first 64 bits of the payload of the original datagram that elicited the error message. As discussed in Section 3, for obvious reasons one should not expect an attacker to include in the packets it sends more information than that required to by the current specifications.

Appendix D. Advice and guidance to vendors

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

NISCC works extensively with government departments and agencies, commercial organizations 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 NISCC, plus NISCC's PGP public key, are available at http://www.uniras.gov.uk/vuls/ .

Appendix E. Changes from previous versions of the draft

E.1. Changes from draft-gont-tcpm-icmp-attacks-05

- Removed RFC 2119 wording to make the draft suitable for publication as an Informational RFC.

- Added additional checks that should be performed on ICMP error

messages (checksum of the IP header in the ICMP payload, and others).

- Added clarification of the rationale behind each the TCP SEQ check

- Miscellaneous editorial changes

E.2. Changes from draft-ietf-tcpm-icmp-attacks-00

- Added references to the specific sections of each of the referenced specifications

- Corrected the threat analysys

- Added clarification about whether the counter-measures violate the current specifications or not.

- Changed text so that the document fits better in the Informational path

- Added an specific section on IPsec (Section 2.3)

- Added clarification and references on the use of ICMP filtering based on the ICMP payload

- Updated references to obsoleted RFCs

- Added a discussion of multipath scenarios, and possible lose in responsiveness resulting from the reaction to hard errors as soft errors (in Section 5.2.3)

- Miscellaneous editorial changes

E.3. Changes from draft-gont-tcpm-icmp-attacks-04

- Added Appendix C

- Added reference to [I-D.iab-link-indications]

- Added stress on the fact that ICMP error messages are unreliable

o Miscellaneous editorial changes

E.4. Changes from draft-gont-tcpm-icmp-attacks-03

o Added references to existing implementations of the proposed counter-measures

o The discussion in Section 4 was improved

o The discussion in Section 5.2.1 was expanded and improved

o The proposed counter-measure for the attack against the PMTUD was improved and simplified

o Appendix B was added

o Miscellaneous editorial changes

E.5. Changes from draft-gont-tcpm-icmp-attacks-02

o Fixed errors in Section 5.2.1

o The proposed counter-measure for the attack against the PMTUD mechanism was refined to allow quick discovery of the Path-MTU

o Appendix A was added so as to clarify the operation of the counter-measure for the attack against the PMTUD mechanism

o Added Appendix D

o Miscellaneous editorial changes

E.6. Changes from draft-gont-tcpm-icmp-attacks-01

o The document was restructured for easier reading

o A discussion of ICMPv6 was added in several sections of the document

o Added Section on Acknowledgement number checking"/>

o Added Section 4.3

o Added Section 7

o Fixed typo in the ICMP types, in several places

o Fixed typo in the TCP sequence number check formula

o Miscellaneous editorial changes

E.7. Changes from draft-gont-tcpm-icmp-attacks-00

o Added a proposal to change the handling of the so-called ICMP hard errors during the synchronized states

o Added a summary of the relevant RFCs in several sections

o Miscellaneous editorial changes

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

Full Copyright Statement

Intellectual Property

ietf-ipr@ietf.org.

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).