TCP Maintenance and Minor Extensions (tcpm)
Internet-Draft
Expires: June 22, 2005

F. Gont
UTN/FRH

December 22, 2004

ICMP attacks against TCP
draft-gont-tcpm-icmp-attacks-03.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

Recently, awareness has been raised about several threats against the TCP [1] protocol, which include blind connection-reset attacks [12]. These attacks are based on sending forged TCP segments to any of the TCP endpoints, requiring the attacker to be able to guess the four-tuple that identifies the connection to be attacked.

While these attacks were known by the research community, they were considered to be unfeasible. However, increases in bandwidth availability, and the use of larger TCP windows [13] have made these attacks feasible. Several general solutions have been proposed to either eliminate or minimize the impact of these attacks [14][15][16]. For protecting BGP sessions, specifically, a counter-measure had already been documented in [17], which defines a new TCP option that allows a sending TCP to include a MD5 [18] signature in each transmitted segment.

All these counter-measures address attacks that require an attacker to send forged TCP segments to the attacked host. However, there is still a possibility for performing a number of attacks against the TCP protocol, by means of ICMP [2]. These attacks include, among others, blind connection-reset attacks.

This document aims to raise awareness of the use of ICMP to perform a number of attacks against TCP, and proposes several counter-measures that can eliminate or minimize the impact of these attacks.

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 [3].

2. Background

2.1 The Internet Control Message Protocol (ICMP)

The Internet Control Message Protocol (ICMP) is used in the Internet Architecture 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 [19].

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 host, to raise awareness of the network problem. In the same way, there are a number of cases in which an end-system may generate an ICMP error message when it finds a problem while processing a datagram. These error messages are notified to the corresponding transport-protocol instance.

When the transport protocol is notified of the error condition, it will perform a fault recovery function. That is, it will try to survive the network failure.

In the case of TCP, the typical fault recovery policy is as follows:

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

- If the network problem being reported is a soft error, just record this information, and repeatedly retransmit the segment until either it gets acknowledged, or the connection times out.

Some stacks honor hard errors only for connections in any of the synchronized states (ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK or TIME-WAIT).

2.1.1 ICMP for IP version 4 (ICMP)

[2] 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 network errors to the sending host.

The Host Requirements RFC [4] 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 hard errors. Thus, any of these ICMP messages could elicit a connection abort.

The ICMP specification also defines the ICMP Source Quench message (type 4, code 0), which is meant to provide a mechanism for flow control and congestion control. The Requirements for IP Version 4 Routers RFC [5], however, states that experience has shown this ICMP message is ineffective for handling these issues.

[6] 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 hosts to determine the MTU of an arbitrary internet path. For obvious reasons, those systems implementing the Path MTU Discovery (PMTUD) mechanism do not treat ICMP error messages of type 3 code 4 as hard errors.

Appendix D of [7] provides information about which ICMP error messages are produced by hosts, intermediate routers, or both.

2.1.2 ICMP for IP version 6 (ICMPv6)

[8] specifies the Internet Control Message Protocol (ICMPv6) to be used with the Internet Protocol version 6 (IPv6) [9].

Even though ICMPv6 didn't exist when [4] was written, one could extrapolate the concept of "hard errors" to ICMPv6 Type 1 (Destination Unreachable) codes 1 (communication with destination administratively prohibited) and 4 (port unreachable). Thus, any of these messages could elicit a connection abort.

ICMPv6 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. For IPv6, intermediate systems do not fragment IP packets. Thus, there is an implicit "don't fragment" bit set in every IPv6 datagram sent on a network. Therefore, hosts do not treat ICMPv6 "Packet Too Big" messages as a hard errors, but use them to discover the MTU of the corresponding internet path, as part of the Path MTU Discovery mechanism for IP Version 6 [10].

Appendix D of [7] provides information about which ICMPv6 error messages are produced by hosts, intermediate routers, or both.

2.2 Handling of ICMP errors

The Host Requirements RFC [4] states that a TCP MUST act on an ICMP error message passed up from the IP layer, directing it to the connection that created the error.

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

Neither the Host Requirements RFC [4] nor the original TCP specification [1] recommend any security checks on the received ICMP messages. Thus, as long as the ICMP payload contains the correct four-tuple that identifies the communication instance, it will be processed by the corresponding transport-protocol instance, and the corresponding action will be performed.

Therefore, an attacker could send a forged ICMP message to the attacked host, and, as long as he is able to guess the four-tuple that identifies the communication instance to be attacked, he can use ICMP to perform a variety of attacks.

As discussed in [12], there are a number of scenarios in which an

attacker may be able to know or guess this four-tuple. Furthermore, it must be noted that most Internet services use the so-called "well-known" ports, so that only the client port would need to be guessed. In the event that an attacker had no knowledge about the range of port numbers used by clients, this would mean that an attacker would need to send, at most, 65536 packets to perform any of the attacks described in this document.

It is clear that security checks should be performed on the received ICMP error messages, to mitigate the impact of the attacks described in this document.

3. ICMP attacks against TCP

ICMP messages can be used to perform a variety of attacks. These attacks have been discussed by the research community to a large extent.

Some TCP/IP implementations have added security checks on the received ICMP error messages to minimize the impact of these attacks. However, as there has not been any official proposal about what would be the best way to deal with these attacks, these security checks have not been widely implemented.

Section 4 of this document discusses the constraints in the general counter-measures that can be implemented against the attacks described in this document. Section 5 proposes several general counter-measures that apply to all the ICMP attacks described in this document. Finally, 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 mitigate their impact. These attack-specific counter-measures are meant to be additional counter-measures to the ones proposed in Section 5. In particular, all TCP implementations SHOULD perform the TCP sequence number checking described in Section 5.1.

4. Constraints in the possible solutions

For ICMPv4, [2] 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. [4] 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 suggesting that implementations may include more data from the original packet than

that required by the original ICMP specification. The Requirements for IP Version 4 Routers RFC [5] 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, that means that the only fields that will be included are: the source port number, the destination port number, and the 32-bit TCP sequence number. This clearly imposes a constraint on the possible security checks that can be performed, as there is not much information avalable on which to perform them. While there exists a proposal to recommend hosts to include more data from the original datagram in the payload of ICMP error messages [20], and some TCP/IP implementations already do this, we cannot yet propose any work-around based on checks performed on any data past the first 64 bits of the payload of the original IP datagram that elicited the ICMP error message. Thus, the only check that can be performed on the ICMP error message is that of the TCP sequence number contained in the payload.

As discussed above, for those ICMP error messages generated by routers, we can expect to receive much more octets from the original packet than just the entire IP header and the first 64 bits of the transport protocol header. Therefore, not only can hosts check the TCP sequence number contained in the payload of the ICMP error message, but they can also perform further checks such as checking the TCP acknowledgement number, as discussed in Section 5.2.

For ICMPv6, 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) [8]. Thus, further checks (as those described above) can be performed on the received ICMP error messages.

5. 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 5.1.

5.1 TCP sequence number checking

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 [21]). 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.

5.2 TCP acknowledgement number checking

As discussed in Section 4, for those ICMP error messages that are generated by intermediate routers, additional checks can be performed. TCP SHOULD check that the TCP Acknowledgement number contained in the payload of the ICMP error message is withing the range SEG.ACK <= RCV.NXT. This means that the TCP Acknowledgement number should correspond to data that have already been acknowledged.

This would reduce the possibility of considering a spoofed ICMP packet as valid by a factor of two.

5.3 Port randomization

As discussed in the previous sections, in order to perform any of the attacks described in this document, an attacker needs to guess (or know) the four-tuple that identifies the connection to be attacked. Randomizing the ephemeral ports used by the clients would make it harder for an attacker to perform any of the attacks discussed in this document.

[22] discusses a number of algorithms to randomize the ephemeral ports used by clients.

Also, a proposal exists to enable TCP to reassign a well-known port number to a random value [23].

5.4 Authentication

Hosts could require ICMP error messages to be authenticated [7], 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 intermediate routers.

[7] contains a discussion on the authentication of ICMP messages.

5.5 Filtering ICMP errors 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. Thus, 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 used as a counter-measure. Systems performing 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.

6. Blind connection-reset attacks

6.1 Description

The Host Requirements RFC [4] states that a host SHOULD abort the corresponding connection when receiving an ICMP error message that indicates a hard error.

Thus, an attacker could use ICMP to perform a blind connection-reset attack. That is, even being off-path, an attacker could reset any TCP connection taking place. In order to perform such an attack, an attacker would send 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.

As discussed in Section 2.2, all an attacker needs to know to perform such an attack is the socket pair that identifies the TCP connection to be attacked. In some scenarios, the IP addresses and port numbers in use may be easily guessed or known to the attacker [12].

Some stacks are known to extrapolate ICMP 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.

There are some points to be considered about this type of attack:

- The source address of the ICMP error message need not be forged. Thus, simple filtering based on the source address of ICMP packets would not serve as a counter-measure against this type of attack.

- Even if TCP itself were protected against the blind connection-reset attack described in [12] and [14], the type of attack described in this document could still succeed.

6.2 Attack-specific counter-measures

6.2.1 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 minimize (or even eliminate) the impact of 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 host 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 (or completely ignore them) if they are meant for connections that are in synchronized states. For TCP, this means one would treat ICMP protocol unreachable error messages as soft errors (or completely ignore them) if they are meant for connections that are in the ESTABLISHED, FIN-WAIT-1, FIN-WAIT-2, CLOSE-WAIT, CLOSING, LAST-ACK or TIME-WAIT states.

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

This error message indicates that the host 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. However, the Host Requirements RFC [4] states that even those transport protocols that have their own mechanism for notifying the sender that a port is unreachable MUST nevertheless accept an ICMP Port Unreachable for the same purpose. For security reasons, it would be fair to treat ICMP port unreachable messages as soft errors (or

completely ignore them) when they are meant for protocols that have their own mechanism for reporting this error condition.

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. 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 them to completely ignore this ICMP error message. On the other hand, and for obvious reasons, those systems implementing the Path-MTU Discovery (PMTUD) mechanism [6] 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. 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 ignore these messages for connections that are in the 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.

Finally, it is important to note that, as discussed in Section 6.1, hosts MUST NOT extrapolate ICMP errors across TCP connections.

6.2.2 Delaying the connection-reset

An alternative counter-measure could be to delay the connection reset. Rather than immediately aborting a connection, a TCP could abort a connection only after an ICMP error message indicating a hard error has been received a specified number of times, and the corresponding data have already been retransmitted more than some

specified number of times.

For example, hosts could abort connections only after a fourth ICMP error message indicating a hard error is received, and the corresponding data have already been retransmitted more than six 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 6.2.1 is more simple to implement and provides increased protection against this type of attack.

7. Blind throughput-reduction attacks

The following subsections discuss a number of attacks that can be performed against TCP to reduce the throughput of a TCP connection. While these attacks do not reset the attacked TCP connections, they may reduce their throughput to such an extent that they may become practically unusable.

7.1 ICMP Source Quench attack

7.1.1 Description

The Host requirements RFC states 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 party. While this would not reset the connection, it would certainly degrade the performance of the data transfer taking place over it.

7.1.2 Attack-specific counter-measures

The Host Requirements RFC [4] states that hosts MUST react to ICMP Source Quench messages by slowing transmission on the connection. However, as discussed in the Requirements for IP Version 4 Routers RFC [5], research seems to suggest ICMP Source Quench is an ineffective (and unfair) antidote for congestion. Thus, we recommend hosts to completely ignore ICMP Source Quench messages.

7.2 ICMP attack against the PMTU Discovery mechanism

7.2.1 Description

When one IP host has a large amount of data to send to another host, 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 [6].

A technique called "Path MTU Discovery mechanism" (PMTUD) lets IP hosts determine the Path MTU of an arbitrary internet path. [6] and [10] 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 host 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 sending host. This message will report the MTU of the constricting hop, so that the sending host reduces the assumed Path-MTU.

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 host. This message will report the MTU of the constricting hop, so that the sending host can reduce the assumed Path-MTU accordingly.

As discussed in both [6] and [10], the PMTUD can be used to attack TCP. An attacker could reduce the throughput of a TCP connection by sending forged ICMP "Destination Unreachable, fragmentation needed and DF set" packets (or their IPv6 counterpart) to the sending host, and making these packets report a low MTU.

For IPv4, this reported Next-Hop MTU could be as low as 68 octets, as [11] 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) [9].

Thus, this attack could considerably readuce the throughput that can

be achieved with the attacked TCP connection.

7.2.2 Attack-specific counter-measures

An analogous counter-measure to that described in Section 6.2.2 could be implemented to greatly minimize the impact of this attack.

For IPv4, this would mean that upon receipt of an ICMP "fragmentation needed and DF bit set" error message, TCP would just record this information, and would honor it only when it had received a specified number of ICMP "fragmentation needed and DF bit set" messages, and provided the corresponding data had already been retransmitted a specified number of times.

For IPv6, the same mechanism would be implemented for handling ICMPv6 "Packet Too Big" error messages.

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) until 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 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 performing Initial Path-MTU Discovery from those performing Path-MTU Update, a new variable should be introduced to TCP: maxsizeacked.

This variable would hold the size (in octets) of the largest packet that has so far been sent and 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.

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.

Upon receipt of an ICMP "Packet Too Big" error message, the Next-Hop MTU claimed by the ICMP message (henceforth "claimedmtu") would be compared with maxsizeacked. If claimedmtu is equal to or larger than maxsizeacked, then TCP is supposed to be at the Initial Path-MTU Discovery stage, and thus the ICMP "Packet Too Big" error message should be honored. That is, the assumed Path-MTU should be updated according to the Next-Hop MTU claimed in the ICMP error message.

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 will therefore delay the update of the assumed Path-MTU.

To perform this delay, two new parameters should be introduced to TCP: MAXPKTTOOBIG, and MAXSEGRTO. MAXPKTTOOBIG would specify the number of times an ICMP "Packet Too Big" must be received before it can be honored to change the Path-MTU. MAXSEGRRTO would specify the number of times a given segment must timeout before an ICMP "Packet Too Big" error message can be honored.

Two variables would be needed to implement the proposed fix: npkttoobig, and nsegrto. npkttoobig would be initialized to zero, and would be incremented by one everytime a valid ICMP "Packet Too Big" error message is received. It would be reset to zero everytime an ICMP "Packet Too Big" error message is honored to change the assumed Path-MTU for given internet path. nsegrto would be initialized to zero, and would be incremented by one everytime the corresponding segment times out.

Thus, the assumed Path-MTU for a given internet path would be changed when an ICMP "Packet Too Big" is received, provided npkttoobig >=

MAXPKTTOOBIG and nsegrto >= MAXSEGRTO. When the assumed Path-MTU is updated, maxsizeacked should be set to claimedmtu, so as to allow the Path-MTU to be discovered quickly in the event the Path-MTU for the connection increases some time later.

The rationale behind this proposed delay is that if there is progress on the connection, the ICMP "Packet Too Big" errors must be a false claim.

MAXPKTTOOBIG can be set to any value greater than or equal to 1. MAXSEGRTO can be set, in principle, to any value greater than or equal to 0.

Setting MAXPKTTOOBIG to 1 and MAXSEGRTO to 0 would make TCP perform the traditional PMTUD mechanism defined in [6] and [10].

When the values chosen for MAXSEGRTO and MAXPKTTOOBIG are such that (MAXSEGRTO - MAXPKTTOOBIG) > 0, it somehow means the implementation is acknowledging that segments may be lost and routers may be rate-limiting their ICMP traffic.

MAXPKTTOOBIG and MAXSEGRTO might be a function of the Next-Hop MTU claimed in the received ICMP "Packet Too Big" message. That is, higher values for MAXPKTTOOBIG and MAXSEGRTO could be imposed when the received ICMP "Packet Too Big" message claims a Next-Hop MTU that is smaller some specified value.

A MAXPKTTOOBIG of 1 and a MAXSEGRTO of 1 should provide enough protection for most cases. In any case, implementations are free to choose higher values for any of these two constants.

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 describes an attack against the PMTUD mechanism that could still succeed, along with a counter-measure against it. However, this attack is unfeasible, and in most cases, non-sensical.

A mechanism that allows hosts to determine the Path-MTU of an arbitrary internet path without the use of ICMP has been is described in [24].

8. Future work

The same considerations discussed in this document for TCP should be applied to other similar protocols.

9. 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.

10. Acknowledgements

This document was inspired by Mika Liljeberg, while discussing some issues related to [25] by private e-mail. The author would like to thank James Carlson, Alan Cox, Juan Fraschini, Markus Friedl, Guillermo Gont, Vivek Kakkar, Michael Kerrisk, Mika Liljeberg, David Miller, Miles Nordin, Eloy Paris, Kacheong Poon, Andrew Powell, and Pekka Savola for contributing many valuable comments.

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

11. References

11.1 Normative References

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

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

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

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

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

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

[7] Kent, S. and R. Atkinson, "Security Architecture for the

Internet Protocol", RFC 2401, November 1998.

[8] Conta, A. and S. Deering, "Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification", RFC 2463, December 1998.

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

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

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

11.2 Informative References

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

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

[14] Stewart, R., "Transmission Control Protocol security considerations", draft-ietf-tcpm-tcpsecure-02 (work in progress), November 2004.

[15] Touch, J., "ANONsec: Anonymous IPsec to Defend Against Spoofing Attacks", draft-touch-anonsec-00 (work in progress), May 2004.

[16] Poon, K., "Use of TCP timestamp option to defend against blind spoofing attack", draft-poon-tcp-tstamp-mod-01 (work in progress), October 2004.

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

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

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

[20] Gont, F., "Increasing the payload of ICMP error messages", (work in progress) draft-gont-icmp-payload-00.txt, 2004.

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

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

[23] Shepard, T., "Reassign Port Number option for TCP", draft-shepard-tcp-reassign-port-number-00 (work in progress), July 2004.

[24] Mathis, M., "Path MTU Discovery", draft-ietf-pmtud-method-03 (work in progress), October 2004.

[25] Gont, F., "TCP's Reaction to Soft Errors", draft-gont-tcpm-tcp-soft-errors-01 (work in progress), October 2004.

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

[27] 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.

[28] Nagle, J., "Congestion control in IP/TCP internetworks", RFC 896, January 1984.

Author's Address

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

Phone: +54 11 4650 8472
EMail: fernando@gont.com.ar

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 5, this Appendix assumes the PMTUD-specific counter-measure is implemented in addition to the TCP sequence number checking described in Section 5.1.

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, 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, all the following subsections assume the TCP implementation at Host 1 has chosen a MAXPKTTOOBIG of 1, and a MAXSEGRTO of 1.

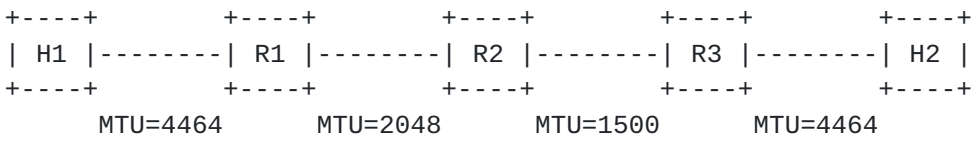

Figure 1: Hypothetical scenario

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

Both npkttoobig and nsegrto are initialized to zero. maxsizeacked is 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 [6] and [10], 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. 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, the Next-Hop MTU reported by the ICMP error message (claimedmtu) is 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.

In line 6, the TCP at H1 sends a segment with 2008 bytes of data, which results in an IP packet of 2048 octets. 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 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.

In line 8, the TCP at H1 sends a segment with 1460 bytes of data, which results in an IP packet of 1500 octets. 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, maxsizeacked is equal to 1500, maxsegrto is equal to 0, and maxpkttoobig 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.

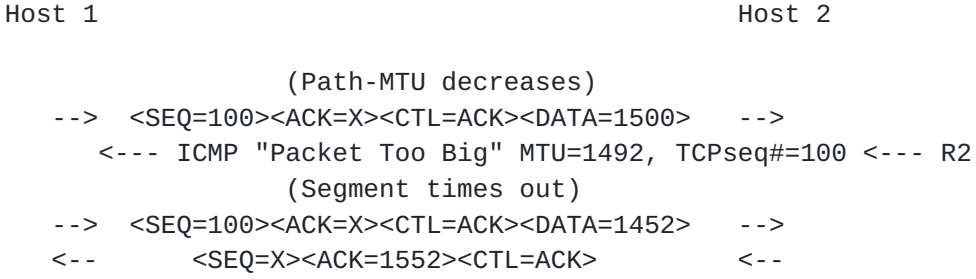

Figure 3: Operation during Path-MTU changes

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 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 maxsizeacked. As claimedmtu is smaller than maxsizeacked, this packet is assumed to be performing Path-MTU Update. Thus, npkttoobig is incremented by one. While npkttoobig is greater than or equal to MAXPKTTOOBIG, nsegrto is still smaller than MAXSEGRTO, and thus the assumed PMTU will not yet be updated.

In line 4, the segment times out. Thus, nsegrto is incremented by 1. As npkttoobig is greater than or equal to MAXPKTTOOBIG and nsegrto is

greater than or equal to MAXSEGRTO, the assumed Path-MTU is updated. npkttoobig and nsegrto are reset to 0, and maxsizeacked is set to claimedmtu.

In line 5, H1 retransmits the data using the updated PMTU. 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. We assume the attacker has guessed the four-tuple that identifies the connection.

```
     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 5.1, 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 maxsizeacked is equal to 1500. Also, npkttoobig must be equal to zero, as no ICMP "Packet Too Big" error messages have been received for the outstanding segments. In the same way, 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, npkttoobig is incremented by one. While npkttoobig is greater than or equal to MAXPKTTOOBIG, nsegrto is still smaller than MAXSEGRTO, and thus the assumed PMTU will not yet be updated.

In line 6, H1 receives an acknowledgement for the segment from line 1, before it times out. At this point, npkttoobig is set back to zero, and the pending ICMP "Packet Too Big" error message is ignored. Therefore, the attack does not succeed.

Appendix B. An attack that could still succeed

This Appendix is for completeness-sake only. The author believes this Appendix will be removed in future revisions of this document. In any case, input on this issue is welcome.

As mentioned in Section 7.2.2, even if the proposed counter-measure for the PMTUD attack were implemented, there is an attack that could still succeed.

Suppose a TCP connection is used by an application which involves the exchange of small amounts of data before large transfers take place. Applications using protocols such as SMTP [26] and HTTP [27], for example, 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: An attack against the PMTUD that could still succeed

Both npkttoobig and nsegrto are initialized to zero. maxsizeacked is 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 sends a small segment to H2. 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 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 will incorrectly honor the ICMP message by updating the assumed MTU.

Thus, the connection will assume a PMTU of 150 octets until the next PMTU-increase "probe" is sent. Depending on the implementation, this "probe" could be sent several minutes later [6][10].

It must be noted that while this attack is theoretically possible, we have assumed the attacker is lucky enough not only to guess the four-tuple that identifies the connection, but also to guess the sequence number of unacknowledged data that are in flight to H2. Given that we assume that only a few small segments are in flight to H2, this is very unlikely. Furthermore, in order to produce any performance impact, the forged ICMP error message should report a Next-Hop MTU that is small enough, but that is larger than maxsizeacked, as TCP would otherwise assume this segment is performing Path-MTU Update, and therefore would delay the update of

the assumed Path-MTU.

Also, it must be noted that algorithms such as that described in [28] will help to avoid sending small segments, and therefore will also help to make maxsizeacked increase quickly, making this attack non-sensical.

In any case, this attack could be completely eliminated by introducing an additional variable: maxsizeinflight. For new connections, this variable would be initialized to the minimum MTU of the internet protocol being used (68 for IPv4, and 1280 for IPv6).

Whenever a packet is to be transmitted, the size of the packet is compared with maxsizeinflight. If the size of the packet to be transmitted is larger than maxsizeinflight, maxsizeinflight is set to that packet size.

Whenever the assumed Path-MTU is updated (either as the result of a segment performing Initial Path-MTU Discovery, or as the result of a segment performing Path-MTU Update), maxsizeinflight is set to the new assumed Path-MTU value.

This way, TCP has a record of the largest packet size it has in flight, and thus can ignore ICMP "Packet Too Big" messages that claim errors that could never have happened.

Appendix C. Advice and guidance to vendors

Vendors are urged to contact NISCC (vulteam@niscc.gov.uk) if they think they will be effected 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 D. Changes from previous versions of the draft

D.1 Changes from draft-gont-tcpm-icmp-attacks-02

- Fixed errors in Section 6.2.1
- The proposed counter-measure for the attack against the PMTUD

mechanism was refined to allow quick discovery of the Path-MTU

- Appendix A was added so as to clarify the operation of the counter-measure for the attack against the PMTUD mechanism
- Added Appendix C
- Miscellaneous editorial changes

D.2 Changes from draft-gont-tcpm-icmp-attacks-01

- The document was restructured for easier reading
- A discussion of ICMPv6 was added in several sections of the document
- Added Section 5.2
- Added Section 5.5
- Added Section 7.2
- Fixed typo in the ICMP types, in several places
- Fixed typo in the TCP sequence number check formula
- Miscellaneous editorial changes

D.3 Changes from draft-gont-tcpm-icmp-attacks-00

- Added a proposal to change the handling of the so-called ICMP hard errors during the synchronized states
- Added a summary of the relevant RFCs in several sections
- Miscellaneous editorial changes

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.