



Results of a Security Assessment of the TCP and IP Protocols and Common Implementation Strategies

Fernando Gont

project carried out on behalf of the
UK CPNI

BSDCan 2009 Conference
May 8-9, 2009, Ottawa, Canada



Agenda (I)

Project overview

Internet Protocol version 4

Identification field

Transmission Control Protocol (TCP)

Overview of basic mechanisms

Security implications of a number of header fields: TCP port numbers, TCP Sequence Number, TCP Window

Security implications of the Urgent mechanism

Security implications of some TCP options: Maximum Segment Size (MSS) option, and Timestamps option

Discussion of some TCP connection-flooding attacks

Discussion of some attacks against TCP send and receive buffers

Discussion of some TCP/IP stack fingerprinting techniques.

Conclusions

Problem Statement (I)

During the last twenty years, many vulnerabilities were found in a number of implementations of the TCP & IP protocols, and in the protocols themselves, which lead to the publication of a number of vulnerability reports by vendors and CSIRTs.

Documentation of these vulnerabilities and the possible mitigations has been spread in a large number of documents.

Some online documentation proposes counter-measures without analyzing their interoperability implications on the protocols. (i.e., wrong and/or misleading advice). See e.g., Silbersack's presentation at BSDCan 2006).

While there had been a fair amount of work on TCP security, the efforts of the security community had never reflected in changes in the corresponding IETF specifications, and sometimes not even in the protocol implementations.

Problem statement (II)

It is very difficult to produce a secure/resilient implementation of the TCP/IP protocols from the IETF specifications.

There was no single document that provided a thorough security assessment of the TCP and IP protocols, and that tried to unify criteria about the security implications of the protocols, and the best possible mitigation techniques.

There was no single document that served as a complement to the official IETF specifications, in the hope of making the task of producing a secure implementation of the protocols easier.

New implementations of the protocols re-implement bugs/vulnerabilities found in older implementations.

New protocols re-implement mechanisms or policies whose security implications have been known from other protocols (e.g., Router Header Type 0 in IPv6 vs. IPv4 source routing).



Project overview

During the last few years, CPNI – formerly NISCC – embarked itself in a project to fill this gap.

The goal was to produce a set of documents that would serve as a security roadmap for the TCP and IP protocols, with the goal of raising awareness about the security implications of the protocols, so that existing implementations could be patched, and new implementations would mitigate them in the first place.

This set of documents would be updated in response to the feedback received from the community.

Finally, we planned to take the results of this project to the IETF, so that the relevant specifications could be modified where needed.

Output of this project

Security Assessment of the Internet Protocol

In July 2008 CPNI published the document “Security Assessment of the Internet Protocol” -- consisting of 63 pages, which include the results of our security assessment of IPv4.

Shortly after, we published the same document as an IETF Internet-Draft (draft-gont-opsec-ip-security-00.txt)

The Internet I-D was finally adopted (by the end of 2008) by the IETF.

Security Assessment of the Transmission Control Protocol (TCP)

In February 2009 CPNI published the document “Security Assessment of the Transmission Control Protocol (TCP)” -- consisting of 130 pages, which include the results of our security assessment of IPv4.

Shortly after, we published the same document as an IETF Internet-Draft (draft-gont-tcp-security-00.txt)

There is currently a very heated debate about this document at the IETF between those that support the idea that the TCP specifications should be maintained/updated, and those who agree that they should be left “as is”.



Internet Protocol version 4



Security Implications of the Identification field

IP Identification field

The IP Identification (IP ID) field is used by the IP fragmentation mechanism.

The tuple {Source Address, Destination Address, Protocol, Identification} identifies fragments that correspond to the same original datagram, and thus the tuple cannot be simultaneously used for more than one packet at any given time.

If a tuple {Source Address, Destination Address, Protocol, Identification} that was already in use for an IP datagram were reused for some other datagram, the fragments of these packets could be incorrectly reassembled at the destination system.

These “IP ID collisions” have traditionally been avoided by using a counter for the Identification field, that was incremented by one for each datagram sent.

Thus, a specific IP ID value would only be reused when all the other values have already been used.



Security implications of the Identification field

If a global counter is used for generating the IP ID values, the IP Identification field could be exploited by an attacker to:

- Infer the packet transmission rate of a remote system

- Count the number of systems behind a NAT

- Perform a stealth port scanning



Randomizing the Identification field

In order to mitigate the security implications of the Identification field, the IP ID should not be predictable, and should not be set as a result of a global counter.

However, it has always been assumed that trivial randomization would be inappropriate, as it would lead to IP ID collisions and hence to interoperability problems.

Some systems (e.g., OpenBSD) have employed PRNG schemes to avoid quick reuse of the IP ID values. However, they have been found to produce predictable sequences.

An analysis of the use of fragmentation for connection-oriented (CO) and for connection-less (CL) protocols can shed some light about which PRNG could be appropriate.



Randomizing the IP ID: CO protocols

Connection-oriented protocols:

The performance implications of IP fragmentation have been known for about 20 years.

Most connection-oriented protocols implement mechanisms for avoiding fragmentation (e.g., Path-MTU Discovery)

Additionally, given the current bandwidth availability, and considering that the IP ID is 16-bit long, it is unacceptable to rely on IP fragmentation, as IP ID values would be reused too quickly regardless of the specific IP ID generation scheme.

We therefore recommend that connection-oriented protocols not rely on IP fragmentation, and they randomize the value they use for the IP Identification field of outgoing segments.

Randomizing the IP ID: CL protocols

Connection-less protocols

They typically lack of:

flow control mechanisms

packet sequencing mechanisms

reliability mechanisms

The scenarios and applications for which they are used assume that:

Applications will be used in environments in which packet-reordering is unlikely.

The data transfer rates will be low enough that flow control is unnecessary

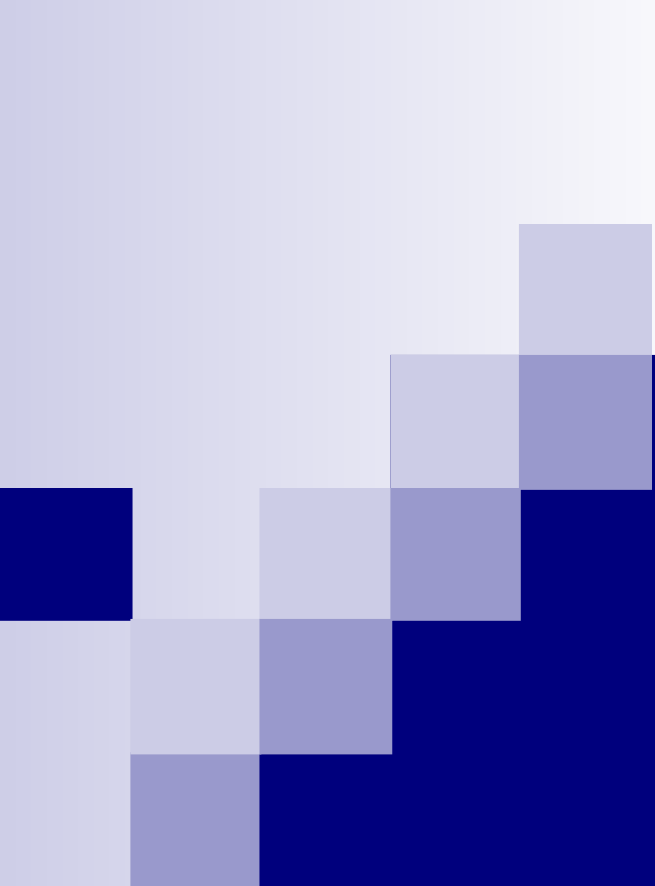
Packet loss is not important and probably also unlikely.

We therefore recommend connection-less protocols to simply randomize the IP ID.

Applications concerned with this policy should consider using a connection-oriented transport protocol.



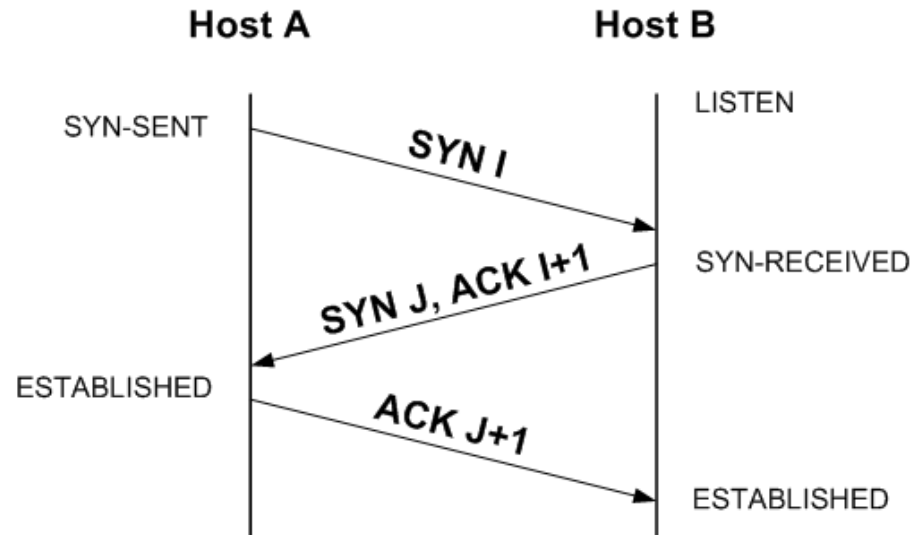
Transmission Control Protocol (TCP)



Overview of the TCP connection-establishment and connection-termination mechanisms

Connection-establishment

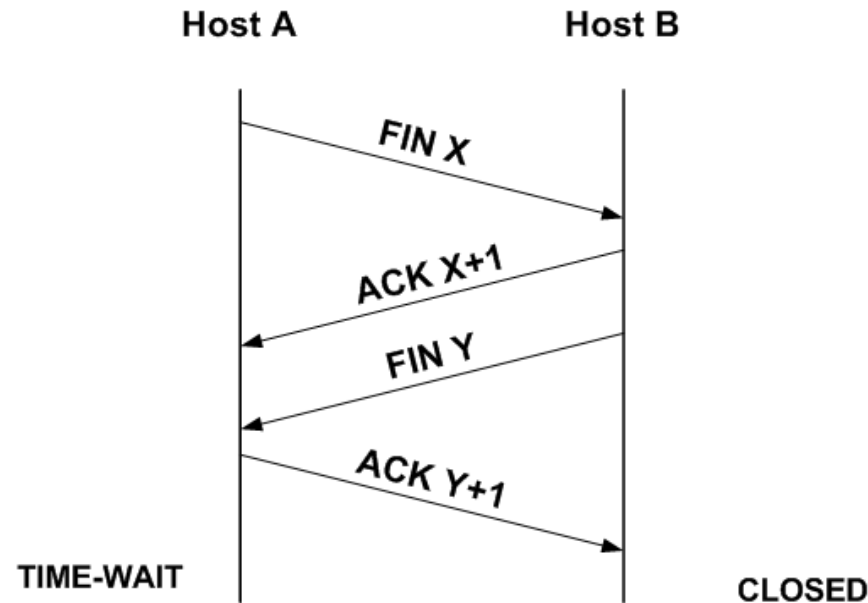
The connection establishment phase usually involves the exchange of three segments (hence it's called "three-way handshake").



When completed, the sequence numbers (and other parameters) will be properly synchronized.

Connection termination

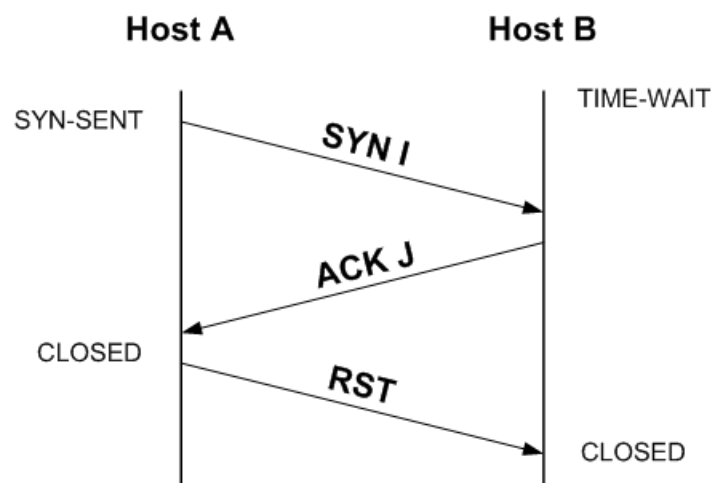
The connection termination phase usually involves the exchange of four segments



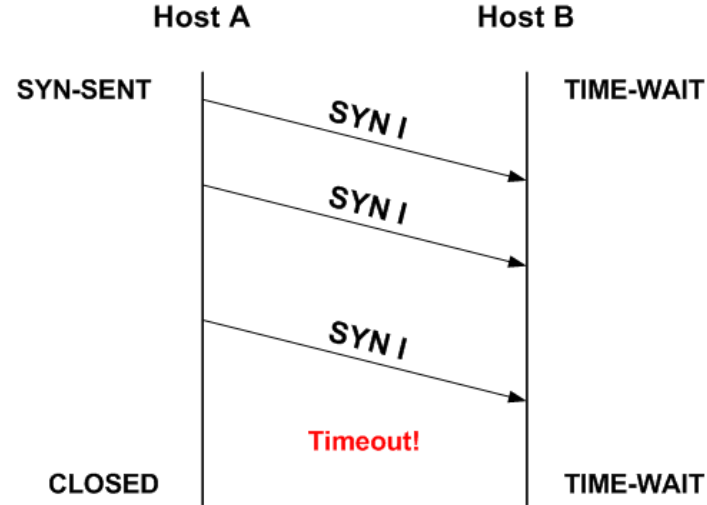
ally stays in the TIME-WAIT state for 4 minutes, while the other end-point moves to the fin

Collision of connection-id's

Due to the TIME-WAIT state, it is possible that when a connection-request is sent to a remote peer, there still exists a previous incarnation of that connection in the TIME-WAIT state. In that scenario, the connection-request will fail.



RFC 793



RFC 1337

It is clear that the collision of connection-id's is undesirable, and thus should be avoided.



Security Implications of a number of TCP header fields



TCP Source Port & Destination Port



TCP port numbers

Trust relationships

While some systems require superuser privileges to bind port numbers in the range 1-1023, no trust should be granted based on TCP port numbers.

Special port numbers

The Sockets API uses port 0 to indicate “any port”. Therefore, a port number of 0 is never used in TCP segments.

Port 0 should not be allowed neither as a source port nor as a destination port.

Ephemeral port range

The IANA has traditionally reserved the range 49152-65535 for the Dynamic and/or Private ports (i.e., the ephemeral ports)

However, different TCP implementations use different port ranges for the ephemeral ports (e.g., 1024-4999, 32768-65535, 49152-65535, etc.)

We recommend TCP implementations to use the range 1024-65535 for the ephemeral ports.



Ephemeral port selection algorithms

When selecting an ephemeral port, the resulting connection-id (client address, client port, server address, server port) must not be currently in use.

If there is currently a local TCB with that connection-id, another ephemeral port should be selected, such that the collision of connection-id's is solved.

However, it is impossible for the local system to actually detect that there is an existing communication instance in a remote system using that connection-id (such as a TCP connection in the TIME-WAIT state).

In the event the selection of an ephemeral port resulted in connection-id that was currently in use at the remote system, a “collision of connectio-id's” would occur.

As a result, the frequency of reuse of connection-id's should be low enough such that collisions of connection-id's are minimized.

TCP Port Randomization

Obfuscation of the TCP ephemeral ports (and hence the connection-id) helps to mitigate blind attacks against TCP connections

The goal is to reduce the chances of an **off-path** attacker from predicting the ephemeral ports used for future connections.

Simple randomization has been found to lead to interoperability problems (connection failures). (See Silbersack's presentation at BSDCan 2006).

A good port randomization algorithm should:

Minimize the predictability of the ephemeral port numbers by an **off-path** attacker.

Avoid quick re-use of the connection-id's

Avoid the use of port numbers that are needed for specific applications (e.g., port 80).

A good port randomization algorithm

The IETF Internet-Draft “Port Randomization” [Larsen, M. and Gont, F., 2008] describes an ephemeral port selection algorithm that’s based on an expression introduced by Steven Bellovin for the selection of ISN’s:

$$\text{Port} = \text{counter} + F(\text{local_IP}, \text{remote_IP}, \text{remote_port}, \text{secret_key})$$

It separates the port number space for connecting to different end-points

It has been found (empyrically) to have better interoperability properties than other obfuscation schemes

It ships with the Linux kernel already.

Sample output of the algorithm

Sample output of the recommended algorithm.

Nr.	IP:port	offset	min_ephemeral	max_ephemeral	next_ephemeral	port
#1	128.0.0.1:80	1000	1024	65535	1024	3048
#2	128.0.0.1:80	1000	1024	65535	1025	3049
#3	170.210.0.1:80	4500	1024	65535	1026	6550
#4	170.210.0.1:80	4500	1024	65535	1027	6551
#5	128.0.0.1:80	1000	1024	65535	1028	3052



TCP Sequence Number

Initial Sequence Numbers (I)

RFC 793 suggests that ISN's must result in a monotonically increasing sequence (e.g., from a global timer), so that the sequence number space of different connections does not overlap. From that point on, it has been assumed that the generation of ISN's such that they are monotonically increasing is key to avoid that corruption in TCP (that could result from "old" segments received for a new connection).

However, protection against old segments is really provided in TCP by two other mechanisms that have nothing to do with the ISN's:

"Quiet time concept": After bootstrapping, a system must refrain from sending TCP segments for $2 \times \text{MSL}$.

TIME-WAIT state: When a TCP connection is terminated, the end-point that performed the "active close" must keep the connection in the TIME-WAIT state for $2 \times \text{MSL}$, thus ensuring that all segments disappear from the network before a new incarnation of the connection is created.

Initial sequence numbers (II)

In the traditional BSD implementation, the ISN generator was initialized to 1 during system boot-strap, and was incremented by 64000 every half second, and by 64000 for each established connection.

Based on the assumption that ISN's are monotonically increasing, BSD implementations introduced some heuristics for allowing quick reuse of the connection-ID's. If a SYN is received for a connection that is in the TIME-WAIT state, then,

If the ISN of the SYN is larger than the last sequence number seen for that direction of the data transfer ($SEG.SEQ > RCV.NXT$), the TCB in the TIME-WAIT state is removed, and another TCP is created in the SYN-RECEIVED state.

Otherwise, the processing rules in RFC 793 are followed.

It is very interesting to note that this hack was motivated by the use of the r^* commands. That is, for short-lived connections, that typically transfer small amounts of data, and/or that typically use a low transfer rate.. Otherwise, these heuristics **fail**.

ISN randomization

The implications of predictable ISN generators have been known for a long time.

ISN obfuscation helps to mitigate blind-attacks against TCP connections.

The goal of ISN obfuscation is to prevent **off-path** attackers from guessing the ISNs that will be used for future connections.

A number of TCP implementations (e.g., OpenBSD) simply randomize the ISN, thus potentially causing the BSD hack to fail. In that scenario, connection failures may be experienced.

We recommend generation of the ISNs as proposed by S. Bellovin in RFC 1948:

ISN = M + F(localhost, localport, remotehost, remoteport, secret)

This scheme produces separates the sequence number space for each connection-id, and generates ISNs that are monotonically-increasing within their respective sequence number spaces.



TCP Window

TCP Window

The TCP Window imposes an upper limit on the maximum data transfer rate a TCP connection can achieve

$$\text{Maximum Transfer Rate} = \text{Window} / \text{Round-Trip Time}$$

Therefore, under ideal network conditions (e.g., no packet loss), the TCP Window should be, at least:

$$\text{TCP Window} \geq 2 * \text{Bandwidth} * \text{Delay}$$

A number of systems and applications use arbitrarily large TCP Windows, in the hope of avoiding the TCP Window from limiting the data transfer rate.

However, larger windows increase the sequence number space that will be considered valid for incoming connections, therefore increasing the chances of an off-path attacker of successfully performing a blind-attack against a TCP connection.

Advice: If an application doesn't require high-throughput (e.g., H.245), use a small window (e.g., 4 KBytes).



TCP Urgent mechanism (URG flag and Urgent Pointer)



Urgent mechanism

The urgent mechanism provide a means for an application to indicate an “interesting point” in the data stream (usually a point in the stream the receiver should jump to). **It is not meant to provide a mechanism for out-of-band (OOB) data.**

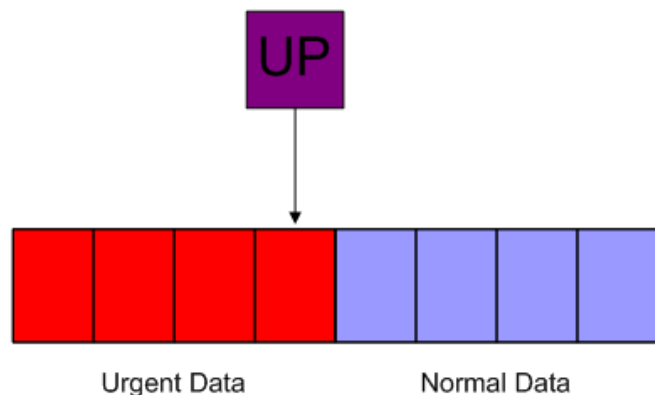
However, most stacks implement the urgent mechanism as out of band data, putting the urgent data in a different queue than normal data.

Ambiguities in the semantics of the UP

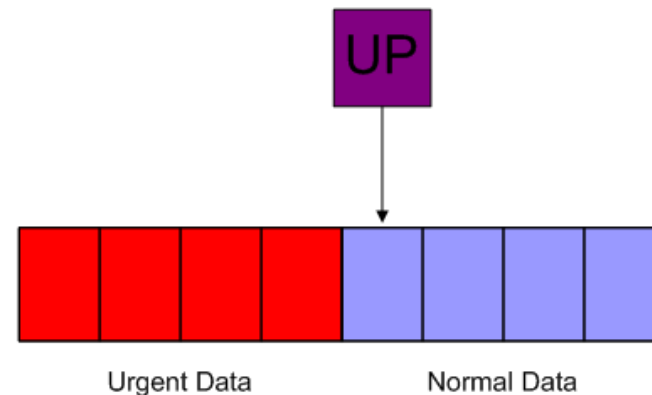
There's a mismatch between the IETF specifications and virtually all real implementations.

“the urgent pointer points to the last byte of urgent data” (IETF) vs. “the Urgent Pointer points to the byte following the last byte of urgent data” (virtually all implementations)

Most implementations nevertheless include a (broken) system-wide toggle to switch between these two possible semantics of the Urgent Pointer



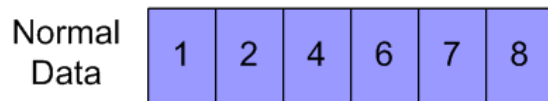
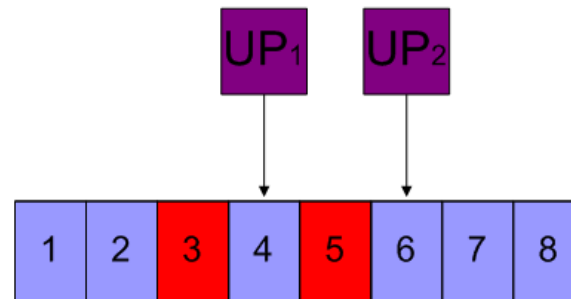
IETF specs
implementations



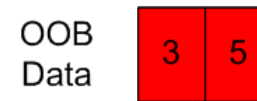
Virtually all

Urgent data as OOB data

TCP/IP stacks differ in how they implement Urgent Data as OOB.
Virtually all stacks only accept a single byte of OOB data
Other stacks (Microsoft's) accept OOB data of any length (*).



Virtually all stacks
stack



Microsoft's

(*). It has been reported that they do not enforce limits on the amount of OOB queued!



Urgent data in the current Internet

Some middle-boxes (e.g., Cisco Pix), **by default**, clear the URG flag and set the Urgent Pointer to zero, thus causing the “urgent data” to become “normal data”.

It is clear that urgent indications are not reliable in the current Internet.

Advice on the urgent mechanism

All the aforementioned issues lead to ambiguities in how urgent data may be interpreted by the receiving TCP, thus requiring much more work on e.g., NIDS.

As discussed before, the urgent mechanism is unreliable in the current Internet (i.e., some widely deployed middle-boxes break it by default).

Advice: Applications should not rely on the urgent mechanism.

If used,

It should be used just as a performance improvement

Applications should set the `SO_OOBINLINE` socket option, so that “urgent data” are processed inline.



TCP Options

MSS (Maximum Segment Size) option

Used to indicate to the remote TCP the maximum segment size this TCP is willing to receive.

Some values are likely to cause undesirable behavior

A value of 0 might cause a connection to “freeze”, as it would not allow any data to be included in the TCP payload.

Other small values may have a performance impact on the involved systems. e.g., they will result in a higher overhead and higher interrupt rate

The security implications of the MSS were first discussed in 2001, but the community never produced any mitigations.

Advice: Sanitize the MSS option as follows:

$$\text{Sanitized_MSS} = \max(\text{MSS}, 536)$$

$$\text{Eff.snd.MSS} = \min(\text{Sanitized_MSS} + 20, \text{MMS_S}) - \text{TCPHdrsize} - \text{IPOptionsize}$$

Timestamps option

TCP timestamps are used to perform Round-Trip Time (RTT) measurement and Protection Against Wrapped Sequence Numbers (PAWS)

For the purpose of PAWS, timestamps are required to be monotonically increasing. However, there's no requirement that the timestamps be monotonically increasing accross TCP connections.

Generation of timestamps such that they are monotonically increasing allows an improved handling of connection-requests (SYN segments) when there's a TCB in the TIME-WAIT state.

Many stacks select the TCP timestamps from a global timer, which is initialized to zero upon system bootstrap.



Security implications of TCP timestamps

Predictable TCP timestamps have a number of security implications:

In order to perform a blind attack against a TCP connection that employs TCP timestamps, an attacker must be able to guess or know the timestamp values in use.

By forging a TCP segment with a timestamp that is larger than the last timestamp received for the target connection, an attacker could cause the connection to freeze.

Therefore, system-wide TCP timestamps are discouraged.

Furthermore, if the timestamps clock is initialized to a fixed value at system bootstrap, the timestamps will leak the system uptime.

Advice on TCP timestamps

Advice: Generate timestamps with a RFC1948-like scheme:

$$\text{timestamp} = T() + F(\text{localhost}, \text{localport}, \text{remotehost}, \text{remoteport}, \text{secret_key})$$

This expression provides a per-destination-endpoint monotonically-increasing sequence, thus enabling the improved handling of SYN segments while avoiding an off-path attacker from guessing the timestamp values used for new connections.

This timestamps generation scheme has been incorporated in Linux
It will most likely be adopted by the IETF in the revision of the TCP timestamps option RFC.



Connection-flooding attacks



Some variants of connection-flooding attacks

SYN-flood: aims at exhausting the number of pending connections for a specific TCP port

Naphta: aims at exhausting the number of ongoing connections

FIN-WAIT-2 flood: aims at exhausting the number of ongoing connections, with connections that are not controlled by a user-space process.

Netkill: aims at exhausting system memory used for the TCP retransmission by issuing a large number of connection requests followed by application requests, and abandoning those connections.

Naphta

The creation and maintenance of a TCP connection requires system memory to maintain shared state between the local and the remote TCPs.

Given that system memory is a limited resource, this can be exploited to perform a DoS attack (this attack vector has been referred to as “Naphta”).

In order to avoid wasting his own resources, an attacker can bypass the kernel implementation of TCP, and simply craft the required packets to establish a TCP connection with the remote endpoint, without tying his own resources.

Counter-measures

Enforcing per-user and per-process limits

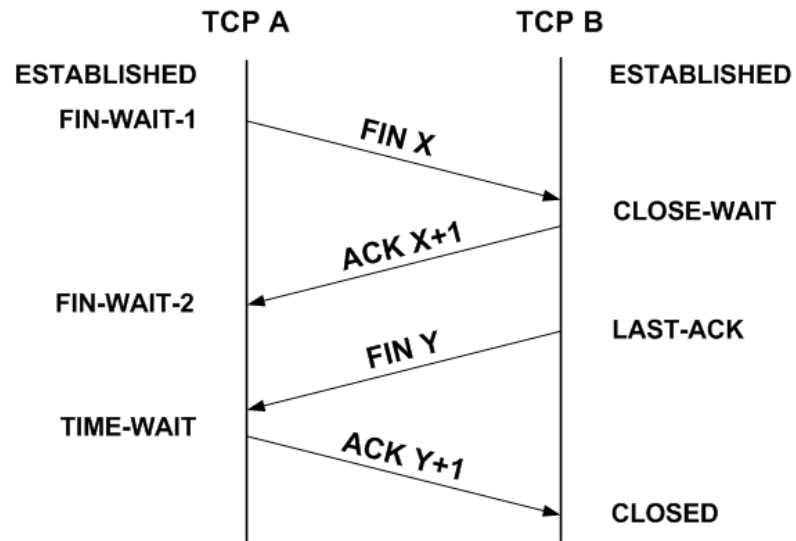
Limiting the number of simultaneous connections at the application

Limiting the number of simultaneous connections at firewalls.

Enforcing limits on the number of connections with no user-space controlling process.

FIN-WAIT-2 flooding attack

A typical connection-termination scenario:



Problems that may potentially arise due to the FIN-WAIT-2 state

There's no limit on the amount of time a connection can stay in the FIN-WAIT-2 state

At the point a TCP gets into the FIN-WAIT-2 state there's no user-space controlling process

Countermeasures for FIN-WAIT-2 flooding

Enforce a limit on the duration of the TIME-WAIT state. E.g., Linux 2.4 enforces a limit of 60 seconds. Once that limit is reached, the connection is aborted.

The counter-measures for the Naptha attack still apply. However, the fact that this attack aims at leaving lots of connections in the FIN-WAIT-2 state will usually prevent an application from enforcing limits on the number of ongoing connections.

Applications should be modified so that they retain control of the connection for most states. This can be achieved by replacing the employing a combination of the `shutdown()`, `setsockopt()`, and `close()`.

TCP should also enforce limits on the number of ongoing connections with no user-space controlling process.



Security implications of the TCP send and receive buffers



TCP retransmission (send) buffer

The **Netkill** attack aims at exhausting the system memory used for the TCP retransmission buffer.

The attacker establishes a large number of TCP connections with the target system, issues an application request, and abandons the aforementioned connections.

The target system will not only waste the system memory used to store the TCB, but also the memory used to queue the data to be sent (in response to the application request).

Counter-measures for the Netkill attack

The countermeasures for the Naphta attack still apply.

In addition, as the malicious connections may end up in the FIN-WAIT-1 state, applications should be modified so that they retain control of the connection for most states. This can be achieved by replacing the employing a combination of the `shutdown()`, `setsockopt()`, and `close()`.

When resource exhaustion is imminent, a connection-pruning policy might have to be applied, paying attention to

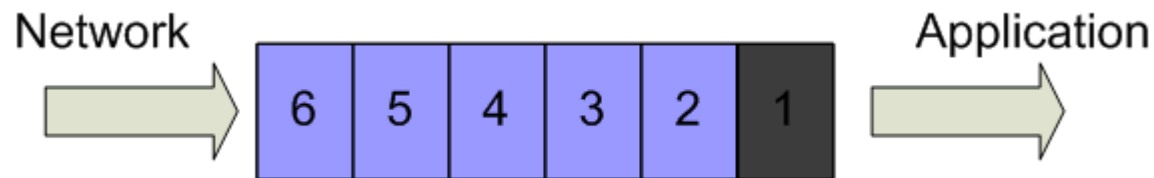
Connections that have advertised a 0-window for a long time

Connections for which the first few windows of data have been retransmitted a large number of times

Connections that fall in one of the previous categories, and for which only a small amount of data have been successfully transferred since their establishment.

TCP reassembly (receive) buffer

When out-of-order data are received, a “hole” momentarily exists in the data stream which must be filled before the received data can be delivered to the application making use of TCP’s services.



This mechanism can be exploited in at least two ways:

An attacker could establish a large number of TCP connections and intentionally send a large amount of data on each of those connections to the receiving TCP, leaving a hole in the data stream so that those data cannot be delivered to the application.

Same as above, but the attacker would send e.g., chunks of one byte of data, separated by holes of e.g., one byte, targeting the overhead needed to hold and link each of these chunks of data.

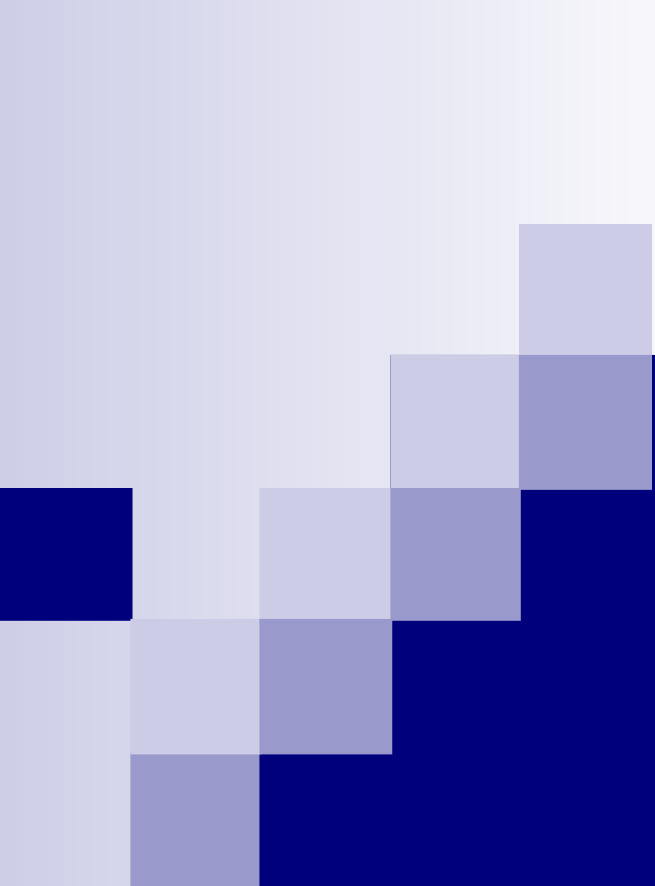


Improvements for handling out-of-order data

TCP implementations should enforce limits on the amount of out-of-order data that is queued at any time.

TCP implementations should enforce limits on the maximum number of “holes” that are allowed for each connection.

If necessary, out-of-order data could be discarded, with no effect on interoperability. This has a performance penalty, though.



Remote Operating System detection via TCP/IP stack fingerprinting



Overview

A number of tools, such as nmap, can perform detect the operating system in use at a remote system, via TCP/IP stack fingerprinting

This is achieved by analyzing the response of the TCP/IP stack to a number of probes that different stack process in different ways

The precision of their results is amazingly good. – It shouldn't be that good!

Question: Wouldn't it be possible for these TCP/IP stacks to respond to most of these probes in exactly the same way?



FIN probe

The IETF specifications leave it unspecified how TCP should respond when a packet that does not have the SYN or ACK bits set is received for a connection that is in the LISTEN state.

Some stacks respond with an RST, while others silently drop the segment.

Advice: reject with an RST those TCP segments that do not have the SYN or ACK bits set and that are received for a connection in the LISTEN state. In all other cases, follow the rules in RFC 793.



Bogus flag test

The attacker sends a TCP segment with at least one of the Reserved bits set. Some stacks ignore this field, while others reset the connection, or reflect the field in the TCP segment sent in response.

Advice: Ignore any flags not supported, and not reflect them if a TCP segment is sent in response to the one just received.

RST sampling

Different implementations differ in the Acknowledgement Number they use in response to segments received for connections in the CLOSED state.

If the ACK bit in the incoming segment is off, the response should be:

$\langle \text{SEQ}=0 \rangle \langle \text{ACK}=\text{SEG.SEQ}+\text{SEG.LEN}+\text{flags} \rangle \langle \text{CTL}=\text{RST, ACK} \rangle$

If the ACK bit in the incoming segment is on, the response should be:

$\langle \text{SEQ}=\text{SEG.ACK} \rangle \langle \text{ACK}=\text{SEG.SEQ}+\text{SEG.LEN}+\text{flags} \rangle \langle \text{CTL}=\text{RST, ACK} \rangle$

That is, the Acknowledgment number should be set to the SEQ of the incoming segment, plus the segment length, and BE incremented by one for each flag that set in the original segment that occupies one byte in the sequence number space.

Port-0 probe

The Sockets API uses port 0 to indicate “any port”. Therefore, a port number of 0 is never used in TCP segments.

Different implementations differ in how they process TCP segments that use 0 as the Source and/or Destination port (e.g., some will allow their use, some will reject incoming connection requests, and some will silently drop the incoming connection requests). This has been exploited for remote OS detection via TCP/IP stack fingerprinting.

Advice: reject with an RST TCP segments that use port number 0 (that do not have the RST bit set).



TCP option ordering

Different TCP implementations enable different options (by default) in their TCP connections.

Additionally, they frame the options differently.

There may be reasons for a TCP to include or not include some specific options. On the other hand, how to frame the options is, for the most part, simply a matter of choice.

More work is needed to get consensus on which options should be included by default, and how to frame them.

An additional benefit resulting from arriving to such consensus is that stacks could implement “TCP option prediction” (i.e., tune the code so that processing of packets with the usual options in the usual order is faster).

Additional fingerprinting techniques

Other parameters can be sampled with the intent to correlate them with specific implementations of TCP:

ISN: While we recommend implementation of the scheme described in RFC1948, some stacks could differ in the granularity of the timer that they use.

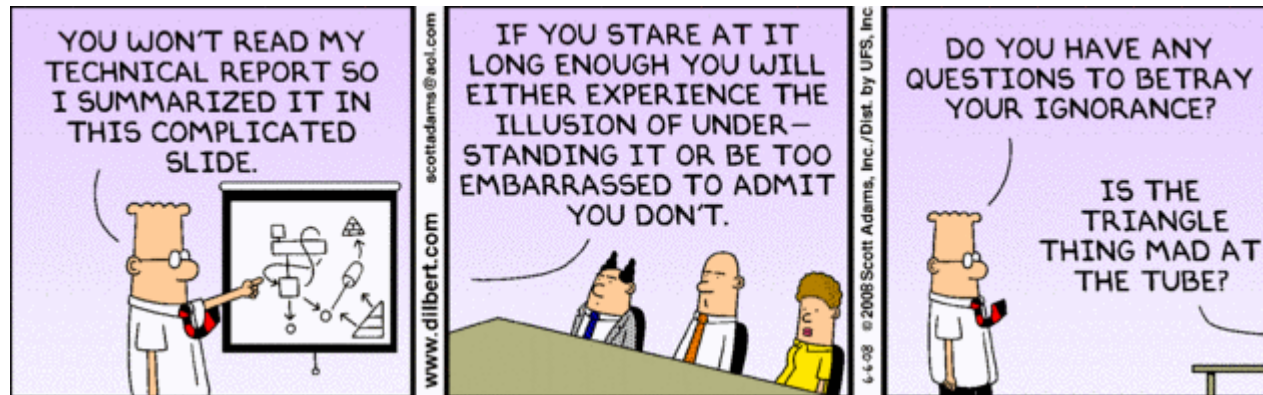
Initial window: Different stacks use different values for the TCP Window advertised in SYN segments. More work is needed to possibly arrive to consensus on the default value to be used.

Retransmission TimeOut (RTO): Different stacks use different values for this parameter. However, of all the fingerprinting techniques, this is the one that is less of a concern, as its precision is highly-dependent on the network conditions.



Conclusions & Further Work

Conclusions and Further Work



Working on TCP/IPv4 security in 2005/2008 probably didn't have much glamour. However, this was something that needed to be done.

Unfortunately, many people will not read past the preface of the documents, but will nevertheless claim that "there's nothing new in this documents".

There seems to be resistance in the IETF to update/fix the specs. – **Get involved!**

We're aware of some efforts in the vendor community to improve the security/resiliency of TCP. Not sure what the end result will be.

Your feedback really matters.



Questions?



Acknowledgements

UK CPNI, for their continued support

Fernando Gont
fernando@gont.com.ar
<http://www.gont.com.ar>