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

Fernando Gont

project carried out on behalf of

UK CPNI

Kernel Conference Australia 2009

Brisbane, Australia, 15-17 July 2009

Agenda

- Overview
- Discussion of some security aspects of IPv4
  - IP Identification field
- Discussion of some security aspects of TCP
  - Overview of some basic TCP mechanisms
  - Port numbers
  - TCP Window
  - TCP urgent mechanism
  - TCP options
  - Connection-flooding attacks
  - Security Implications of the TCP reassembly buffer
  - Remote OS detection via TCP/IP stack fingerprinting
- Conclusions

Overview

(or “why we did what we did”)

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.
- Documentation of these issues and of possible mitigations has been spread among a number of vulnerability reports issued by vendors and CSIRTs and a variety of online documents.
- Some of this documentation proposes counter-measures for these issues without analyzing their interoperability implications on the protocols. (See e.g., Silbersack's presentation at BSDCan 2006).
- The efforts of the security community never resulted in changes in the corresponding IETF specifications, and sometimes not even in the protocol implementations.

Problem Statement (II)

- It was very difficult to produce a secure/resilient implementation of the TCP/IP protocols from the IETF specifications.
- It was painful to spot the correct advice among all the available documentation.
- As a result,
  - New implementations of the protocols re-implemented bugs/vulnerabilities found in older implementations.
  - **New protocols** re-implemented mechanisms or policies whose security implications had 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 and providing advice to mitigate them.
- This set of documents would be updated in response to the feedback received from the comunity.
- Finally, we planned to take the results of this project to the IETF, so that the relevant specifications could be modified where needed.

Ouput of this project

- "Security Assesment of the Internet Protocol"
  - 63-page document, published by the UK CPNI in July 2008.
  - Available at: http://www.cpni.gov.uk/Docs/InternetProtocol.pdf
  - Currently adopted by the IETF as a work item of the opsec wg (draft-ietf-opsec-ip-security)
- **"Security Assessment of the Transmission Control Protocol (TCP)"**
  - 130-page document, published by the UK CPNI in February 2009.
  - Available at: http://www.cpni.gov.uk/Docs/tn-03-09-security-assessment-TCP.pdf
  - Submitted to the IETF (draft-gont-tcp-security), with the IETF currently deciding whether to adopt this document as a wg item of the TCPM working group.

Internet Protocol version 4

IPv4 Identification field

IP IDentification field

- The IP Identification (IP ID) field is used by the IP framentation 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:
  - Count the number of systems behind a NAT
  - Infer the packet transmission rate of a remote system
  - 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 (e.g., it 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 have employed specific PRNG schemes to avoid quick reuse of the IP ID values. However, some of them 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 how to set the IP Identification field.

Randomizing the IP ID: CO protocols

- Most connection-oriented protocols implement mechanisms for avoiding fragmentation (e.g., Path-MTU Discovery)
  - The performance implications of IP fragmentation have been known for about 20 years.
  - Also, 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 quikly regardless of the specific IP ID generation scheme.
- We therefore recommend that connection-oriented protocols not rely on IP fragmentation, and that they randomize the value they use for the IP Identification field of outgoing packets.

Randomizing the IP ID: CL protocols

- Connection-less transport protocols typically lack of:
  - packet sequencing mechanisms
  - flow control 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 and/or the total amount of data to be transferred will be small 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.

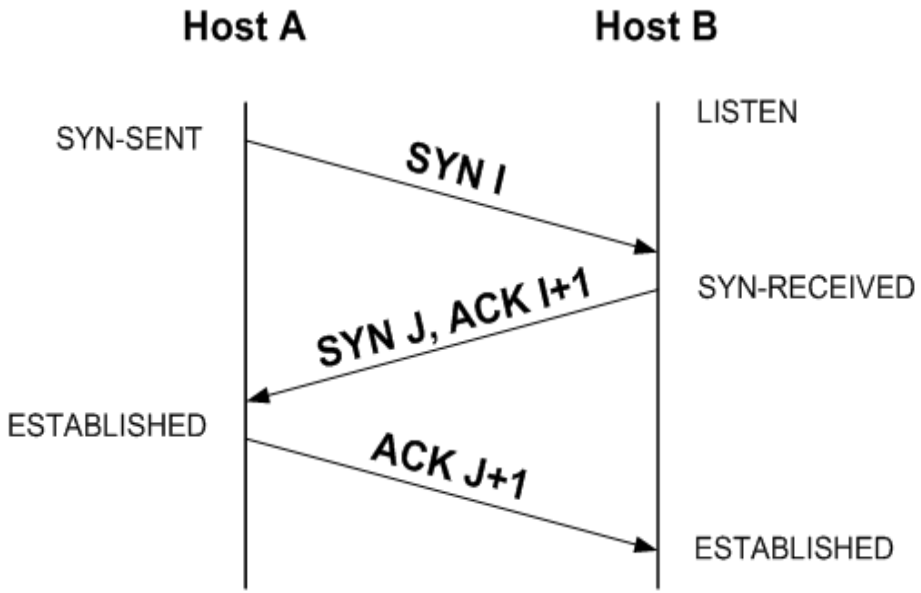
Transmission Control Protocol

Overview of basic TCP mechanisms

(connection establishment & termination)

Connection-establishment

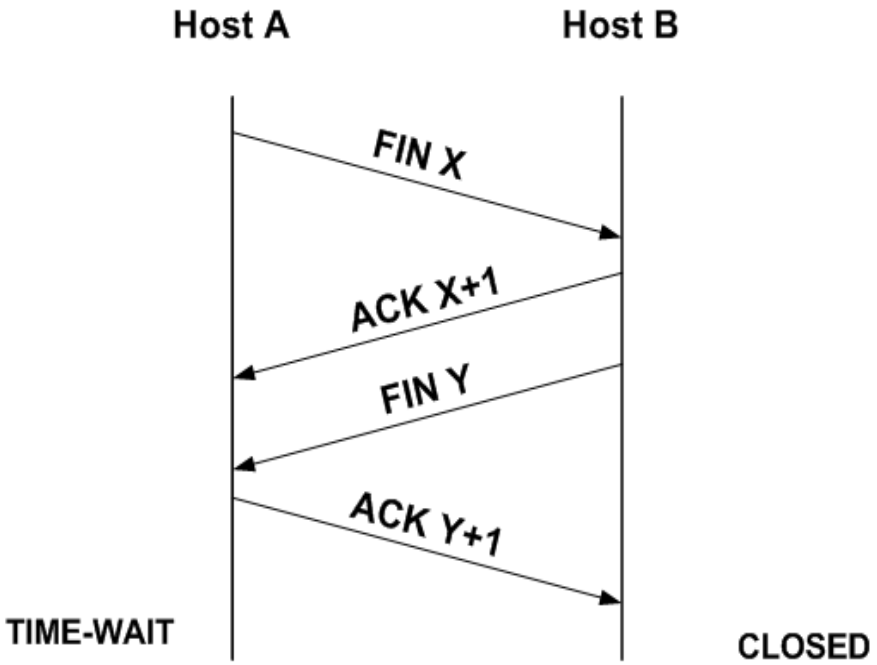
- The connection establishment phase usually involves the exchange of three segments (hence it's called "three-way handshake).



- Once the three-way handshake has completed, the sequence numbers (and other parameters) will be properly synchronized, and the data transfer can proceed.

Connection termination

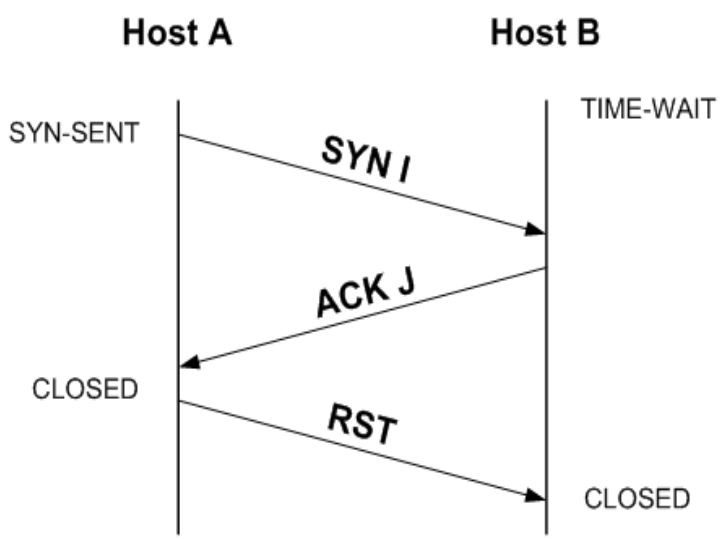
- The connection termination phase usually involves the exchange of four segments



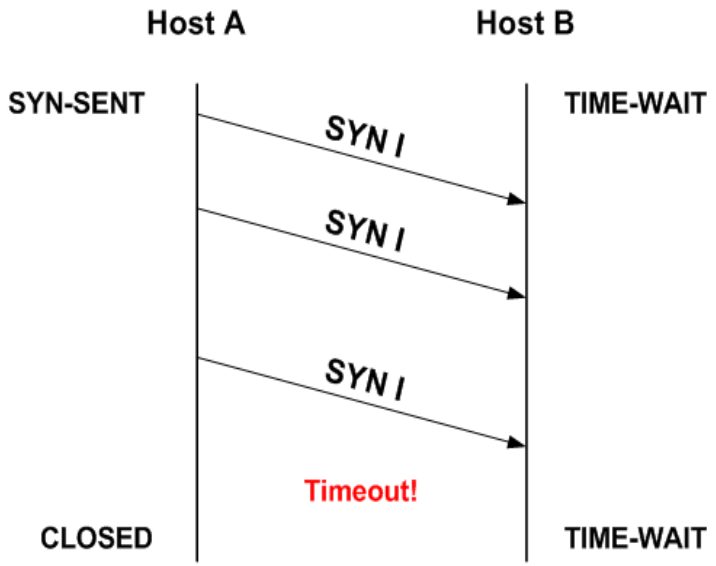
- The TCP that begins the connection-termination phase (Host A) usually stays in the TIME-WAIT state for 4 minutes, while the other end-point moves to the fictional CLOSED state (i.e., it does not keep any state for this connection)

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 collission of connection-id's is undesirable, and thus should be avoided.

TCP port numbers

(port obfuscation)

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 very difficult 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 connection-id's" would occur.
- Most systems have traditionally avoided these collisions by selecting ephemeral ports from a global counter (i.e., a similar approach to that used for the IP Identification field), thus leading to predictable ephemeral ports.

TCP Ephemeral Port Obfuscation

- Predictable ports numbers have negative security implications, as they make it easy for an attacker to guess/predict the connection-id of a target connection.
- However, simple randomization has been found to lead to interoperability problems (connection failures). (See Silbersack's presentation at BSDCan 2006).
- A good port obfuscation scheme 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 TCP port obfuscation 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:

  Port = counter + F(local_IP, remote_IP, remote_port, secret_key)

- It separates the port number space used 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.
- We recommend the implementation of this algorithm (or its improved double-hash variant) for the selection of ephemeral ports.

Sample output of the algorithm

- Sample output of the recommended algorithm.

Nr.	IP:port	F()	min_ephlemeral	max_ephemeral	counter	port
#1	128.0.0.1:80	1000	1024	65535	0	2024
#2	128.0.0.1:80	1000	1024	65535	1	2025
#3	170.210.0.1:80	4504	1024	65535	2	5530
#4	170.210.0.1:80	4504	1024	65535	3	5531
#5	128.0.0.1:80	1000	1024	65535	4	2028

TCP Window

TCP Window

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

  Maximum Transfer Rate = Window / Round-Trip Time

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

  TCP Window >= 2 * Bandwidth * 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

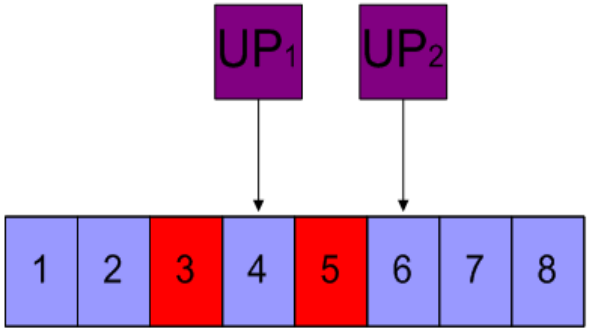
(URG flag and Urgent Pointer)

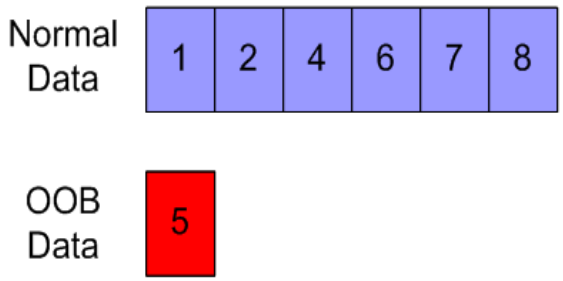
Urgent mechanism

- 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.

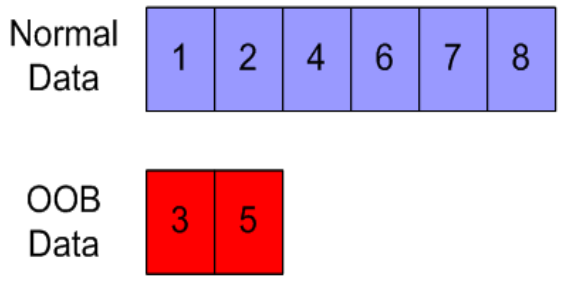
Urgent data as OOB data

- 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

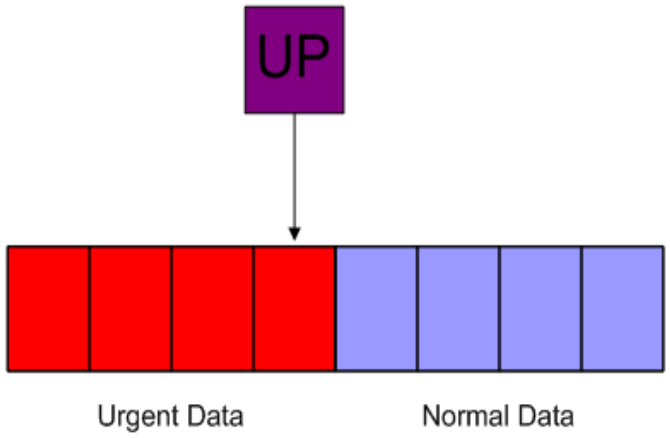


Microsoft's stack

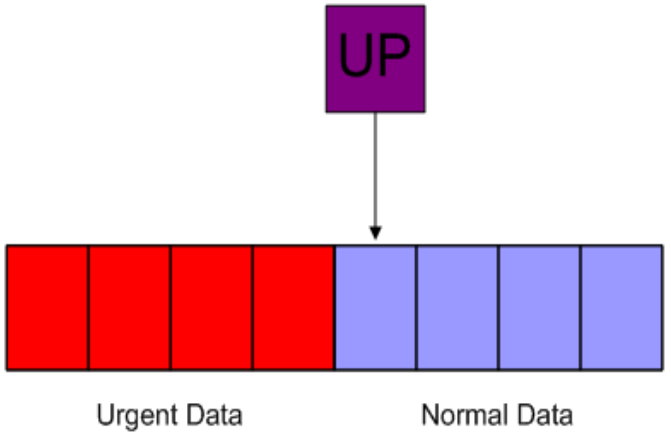
(*) It has been reported that they do not enforce limits on the amount of OOB queued!

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



Virtually all implementations

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 procesed inline.
- The TCPM wg is currently working on a Standards-track document (draft-ietf-tcpm-urgent-data) that discourages the use of the TCP urgent mechanism, updates RFC 1122 to match implementations, and encourages the use od the SO_OOBINLINE socket options when the urgent mechanism is employed.

TCP Options

(TCP Timestamps)

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 accross conections allows an improved handling of connection-requests (SYN segments) when there's a TCB in the TIME-WAIT state (i.e., accept the incomning SYN if it contains a timestamps that is larger than the last timestamp seen in the previous incarnation of the same connection).
- Many stacks select the TCP timestamps from a global timer, which is initialized to zero upon system bootstrap. However, this policy leads to predictable TCP timestamps.

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.
  - Furthermore, if the timestamps clock is initilized to a fixed value at system bootstrap, the timestamps will leak the system uptime.
- Therefore, predictable TCP timestamps should be avoided.
- Some systems (e.g. OpenBSD) randomize the TCP timestamps. However, this prevents the quick reuse of connection-id's.

Advice on TCP timestamps

- Advice: Generate timestamps with a RFC1948-like scheme:

  timestamp = T() + F(localhost, localport, remotehost, remoteport, 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 ships with Linux.
- It will most likely be adopted by the IETF in the revision of the TCP timestamps RFC (RFC 1323).

Connection-flooding attacks

(Naphta and FIN-WAIT-2)

Some variants of connection-flooding attacks

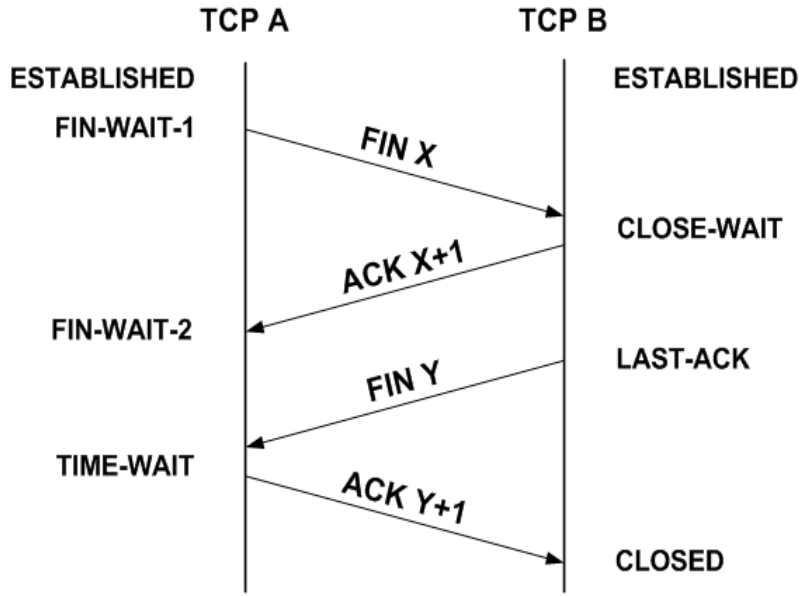
- Naphta: aims at performing a DoS by establishing lots of TCP connections that are then “abandoned”.
- FIN-WAIT-2 flood: aims at performing a DoS by establishing lots of TCP connections and abandoning them in the FIN-WAIT-2 state.

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.

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

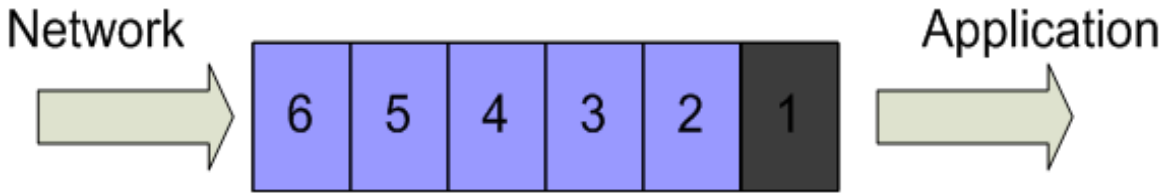
Countermeasures for FIN-WAIT-2 flooding

- Enforce a limit on the duration of the FIN-WAIT-2 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 with a conbination of the shutdown(), setsockopt(), and close().
- TCP should also enforce limits on the number of ongoing connections with no controlling process.

TCP reassembly buffer

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 are 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 OS detection

(via TCP/IP stack fingerprinting)

Remote OS detection

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

Some fingerprinting probes

- Nmap and similar tools send a number of probe packets to the target system to fingerprint its TCP/IP stack (e.g., FIN probe, Bogus flag probe, etc.).
- Different systems respond to these probe packets in different ways.
- We have performed an analysis of each of these fingerprinting techniques, and provided advice on how to respond to each of these probe packets.
- We expect that in the long term remote OS detection based on these probes will have much less precision.

TCP option ordering

- Another important technique for remote OS detection is to fingerprint the TCP options used by the target system.
- 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).

Conclusions

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 these documents" and/or "there's nothing to be done about this".
- Still in 2009, there's lots of work to do to improve the available TCP implementations.
- 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.
- There is some resistance in the IETF to update/fix the specs (talk about politics). – **Get involved!**
- **Your feedback really matters.**

Questions?

Acknowledgements

- UK CPNI, for their continued support
- KCA 2009 organizers, for their support in this conference.

Fernando Gont

fernando@gont.com.ar

http://www.gont.com.ar