TCP Idle Scans in IPv6

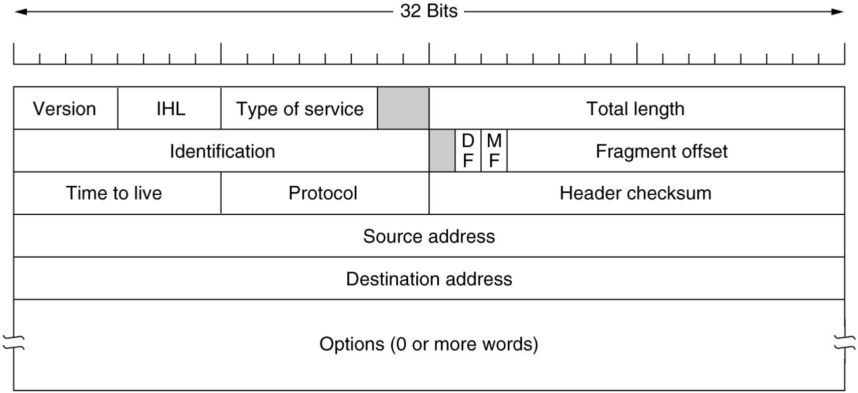
Mathias Morbitzer

Radboud University Nijmegen

Fox-IT

October 16th, 2013

IPv4 Header



32 Bits

Version | IHL | Type of service | Total length

Identification | DF | MF | Fragment offset

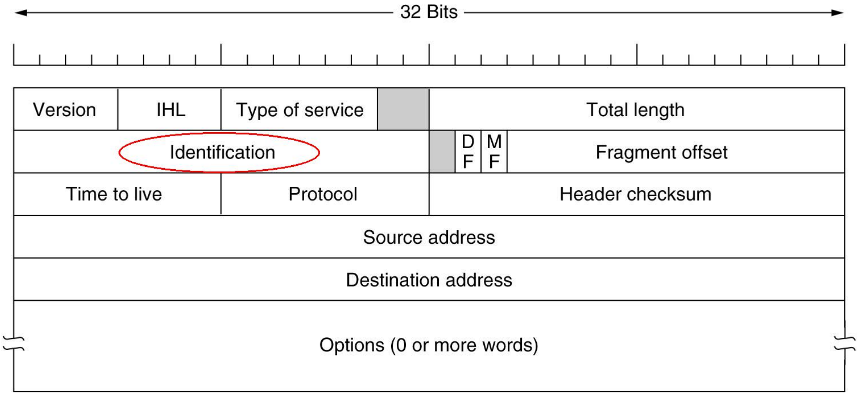
Time to live | Protocol | Header checksum

Source address

Destination address

Options (0 or more words)

IPv4 Header



32 Bits

Version

IHL

Type of service

Total length

Identification

DF

MF

Fragment offset

Time to live

Protocol

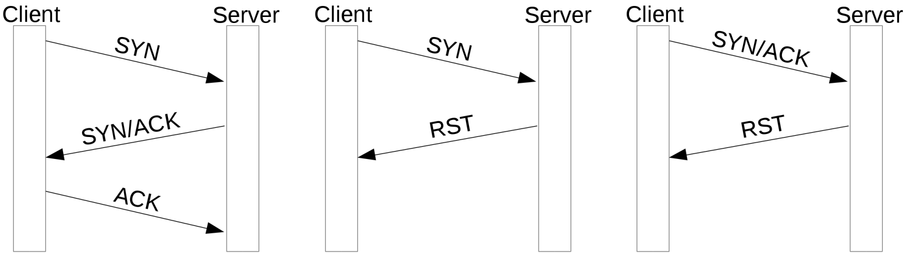
Header checksum

Source address

Destination address

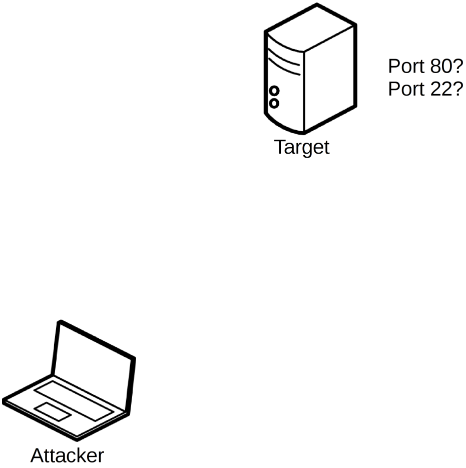
Options (0 or more words)

TCP three way handshake

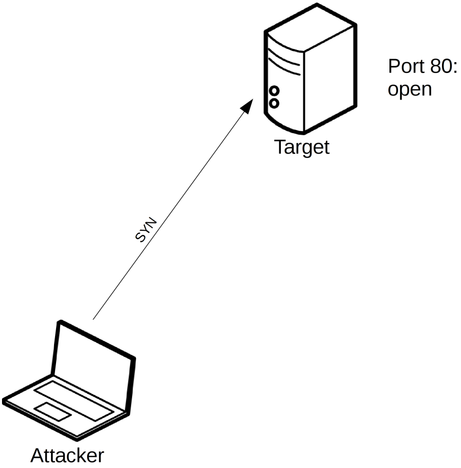


(a) Successful (b) Unsuccessful (c) Unexpected

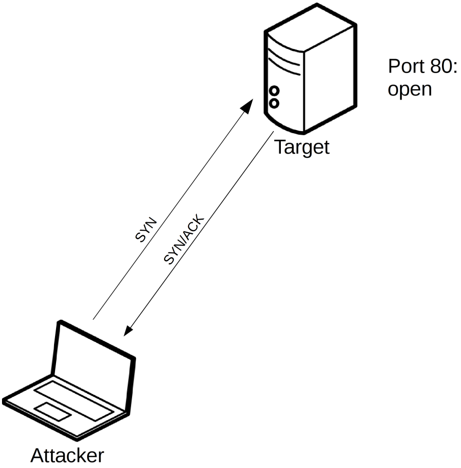
Port scanning



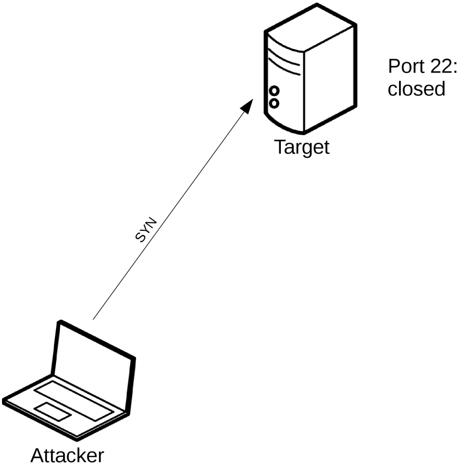
Port scanning



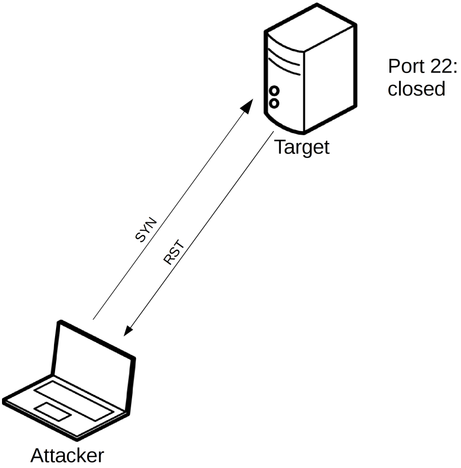
Port scanning



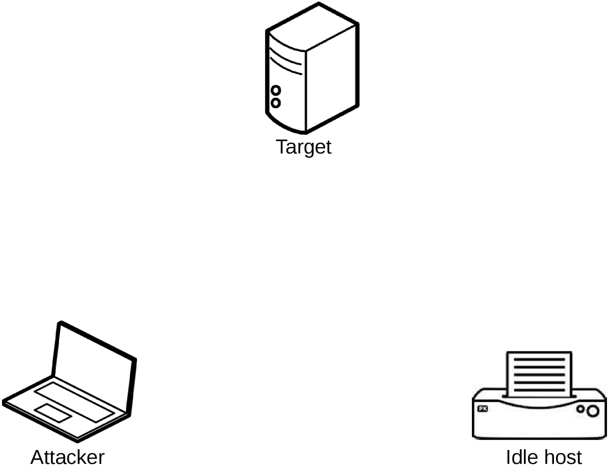
Port scanning



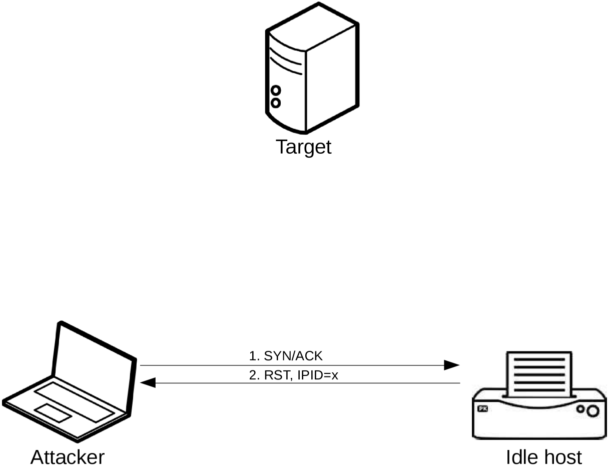
Port scanning



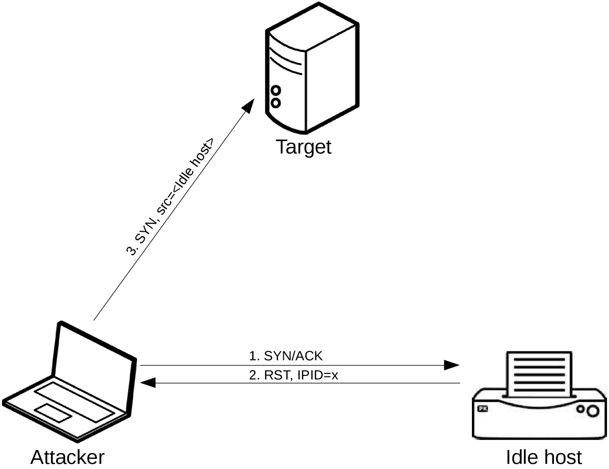
TCP Idle Scan



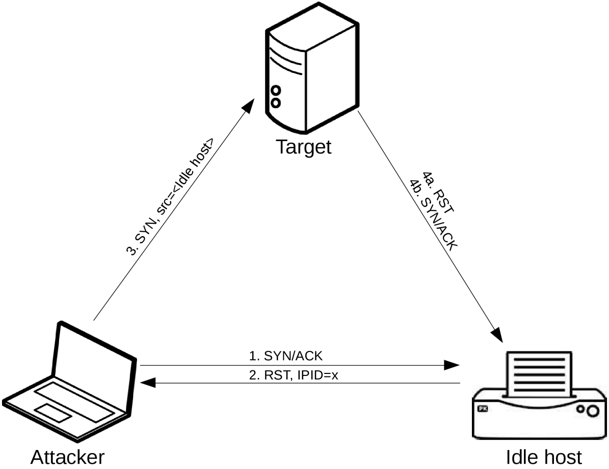
TCP Idle Scan



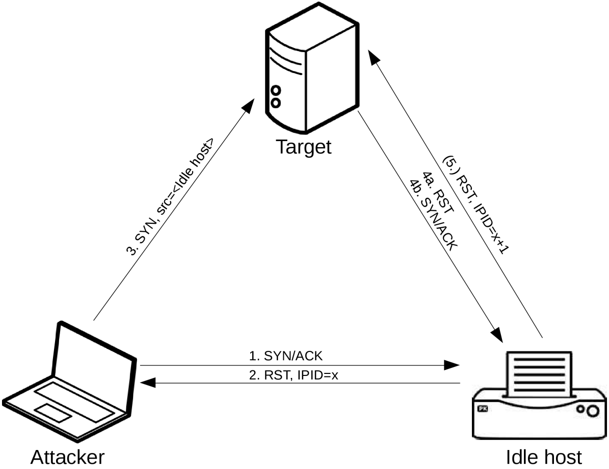
TCP Idle Scan



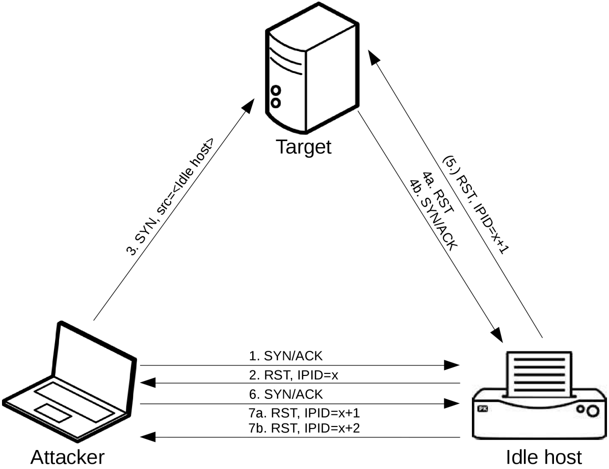
TCP Idle Scan



TCP Idle Scan



TCP Idle Scan



Target

Attacker

Idle host

3. SYN, src=<Idle host>

4a. RST
4b. SYN/ACK

(5.) RST, IPID=x+1

1. SYN/ACK

2. RST, IPID=x

6. SYN/ACK

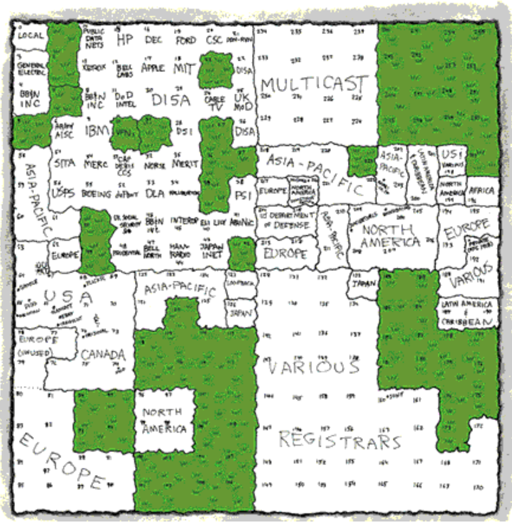
7a. RST, IPID=x+1
7b. RST, IPID=x+2

Requirements for Idle host in IPv4

1. Predictable, global assignment of Identification value

2. Remain idle

Map of the Internet - The IPv4 space, 2006



IPv6

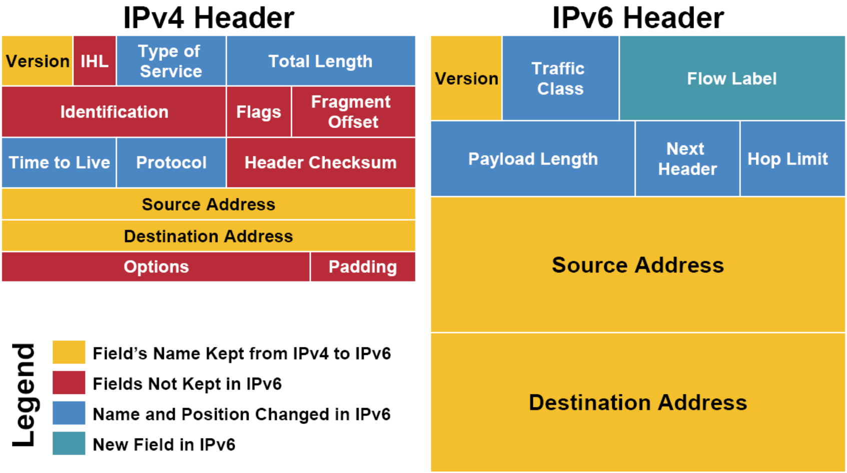
- 128bit addresses instead of 32bit

IPv6

- 128bit addresses instead of 32bit

→ 340 undecillion, 282 decillion, 366 nonillion, 920 octillion, 938 septillion, 463 sextillion, 463 quintillion, 374 quadrillion, 607 trillion, 431 billion, 768 million, 211 thousand and 456 addresses

IPv4 vs IPv6



IPv4 Header

Version | IHL | Type of Service | Total Length

Identification | Flags | Fragment Offset

Time to Live | Protocol | Header Checksum

Source Address

Destination Address

Options | Padding

IPv6 Header

Version | Traffic Class | Flow Label

Payload Length | Next Header | Hop Limit

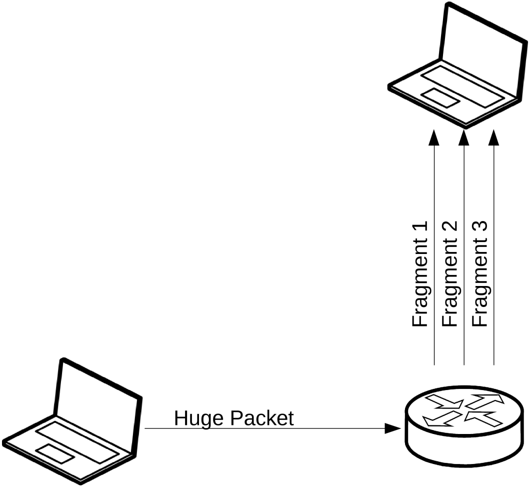
Source Address

Destination Address

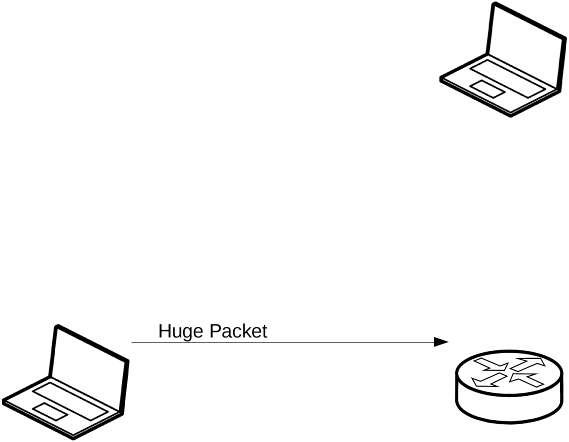
Legend

- 🟨 Field's Name Kept from IPv4 to IPv6
- 🟥 Fields Not Kept in IPv6
- 🟦 Name and Position Changed in IPv6
- 🟩 New Field in IPv6

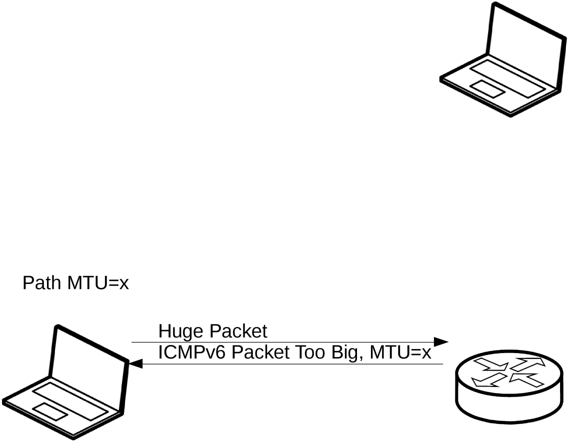
Fragmentation in IPv4



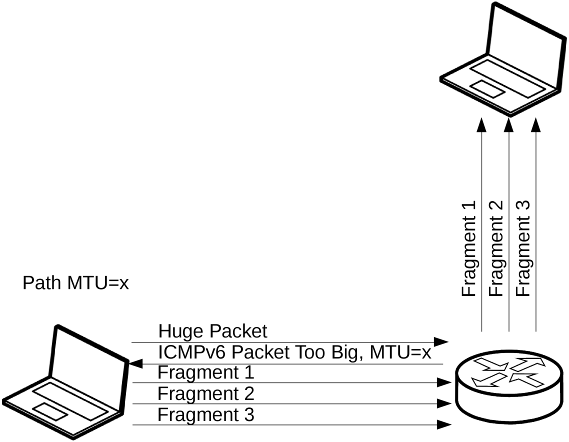
Fragmentation in IPv6



Fragmentation in IPv6

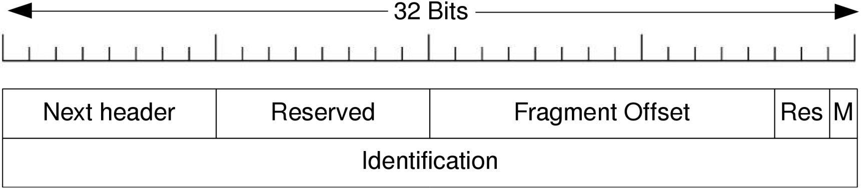


Fragmentation in IPv6



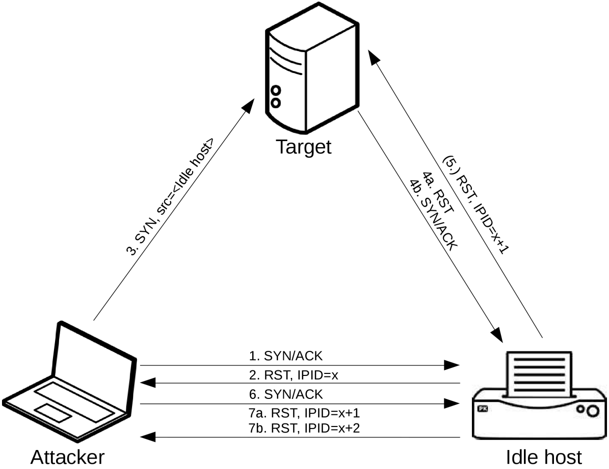
Fragmentation in IPv6

- Extension header used when needed
- Located between IPv6 and TCP header
- Extension header for fragmentation / Fragmentation header:

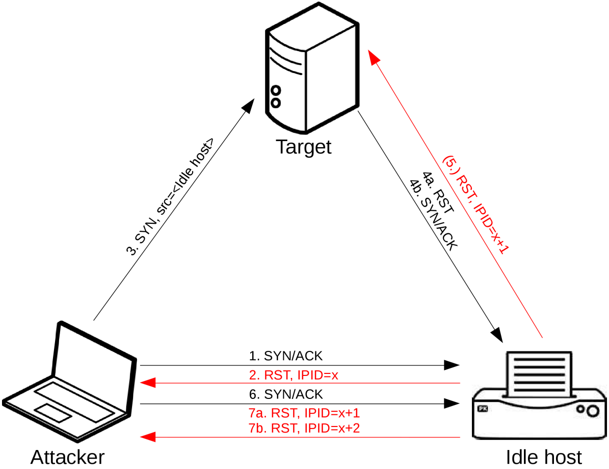


Next header	Reserved	Fragment Offset	Res	M
Identification				

Extension header in all steps?



Not in all...


Target
3. SYN, src=<Idle host>
4a. RST
4b. SYN/ACK
(5.) RST, IPID=x+1
1. SYN/ACK
2. RST, IPID=x
6. SYN/ACK
7a. RST, IPID=x+1
7b. RST, IPID=x+2
Attacker
Idle host

Forcing fragmentation in steps 2 and 7

- Directly participating in the conversation
- Something where we send a lot, and get a lot back

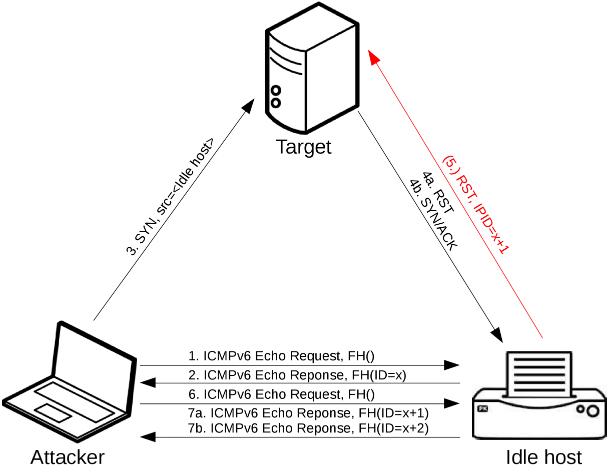
Forcing fragmentation in steps 2 and 7

- Directly participating in the conversation
- Something where we send a lot, and get a lot back
  → How about pings?

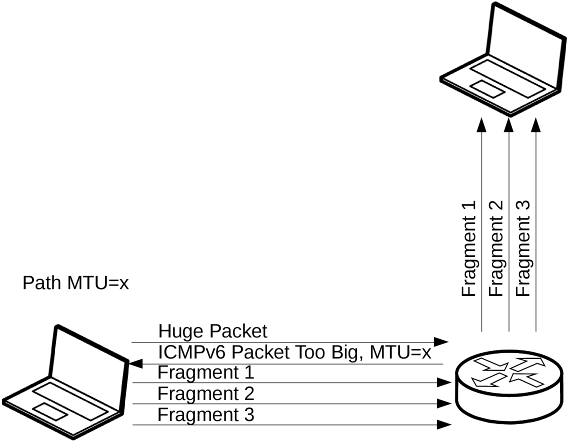
> *The data received in the ICMPv6 Echo Request message MUST be returned entirely and unmodified in the ICMPv6 Echo Reply message. (RFC 4443, ICMPv6)*

- If the Request is fragmented, the Reply will be fragmented too

Forcing fragmentation in steps 2 and 7


Target
3. SYN, src=<Idle host>
4a. RST
4b. SYN/ACK
(5.) RST, IPID=x+1
1. ICMPv6 Echo Request, FH()
2. ICMPv6 Echo Reponse, FH(ID=x)
6. ICMPv6 Echo Request, FH()
7a. ICMPv6 Echo Reponse, FH(ID=x+1)
7b. ICMPv6 Echo Reponse, FH(ID=x+2)
Attacker
Idle host

Forcing fragmentation in step 5



Forcing fragmentation in step 5

- So we can manipulate another host's Path MTU!
- minimum IPv6 MTU: 1280 bytes
- IPv6 + TCP header max 60 bytes
- Let's have a look at RFC 1981

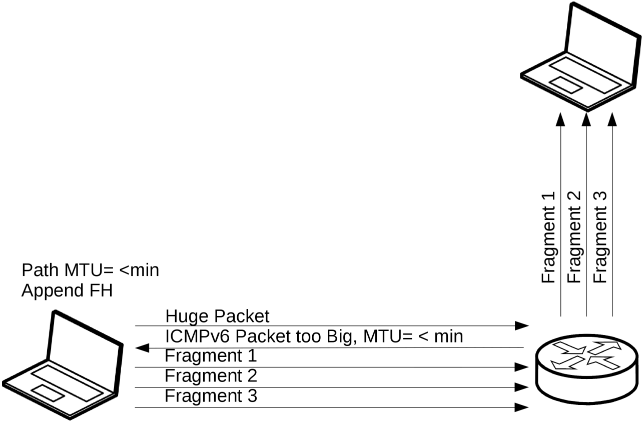
Forcing fragmentation in step 5

When a node receives a Packet Too Big message, it MUST reduce its estimate of the PMTU for the relevant path, based on the value of the MTU field in the message

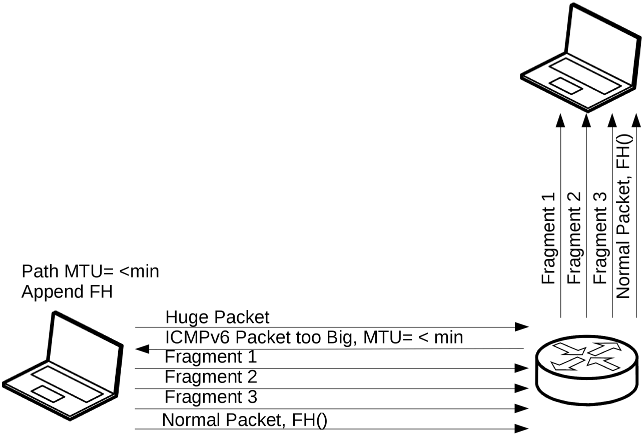
A node MUST NOT reduce its estimate of the Path MTU below the IPv6 minimum link MTU. Note: A node may receive a Packet Too Big message reporting a next-hop MTU that is less than the IPv6 minimum link MTU. In that case, the node is not required to reduce the size of subsequent packets sent on the path to less than the IPv6 minimum link MTU, but rather must include a Fragment header in those packets

(RFC 1981, Path MTU Discovery for IP version 6)

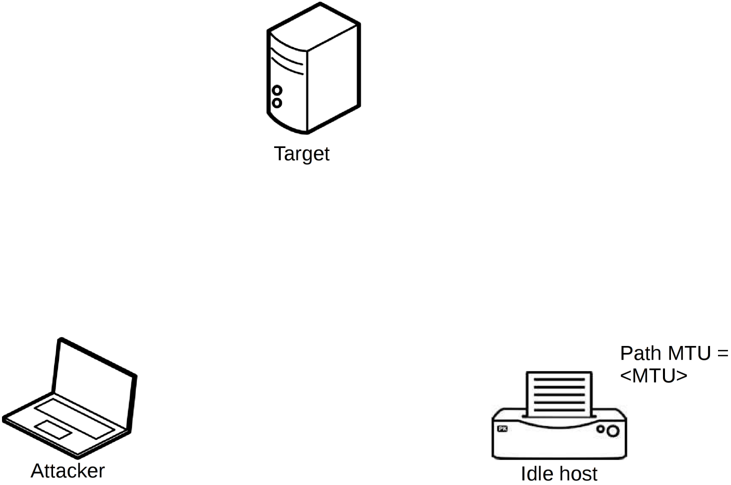
Forcing fragmentation in step 5



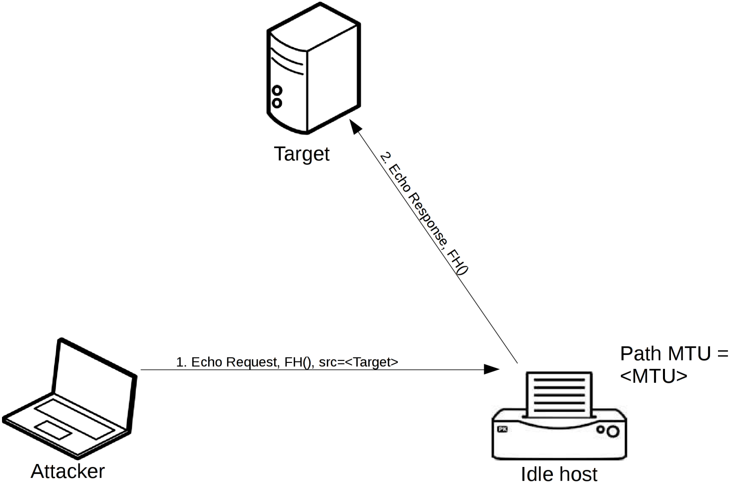
Forcing fragmentation in step 5



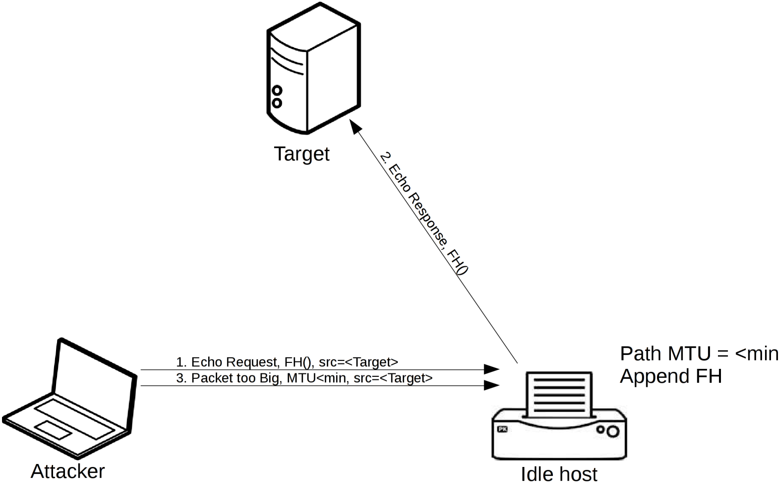
The TCP Idle Scan in IPv6



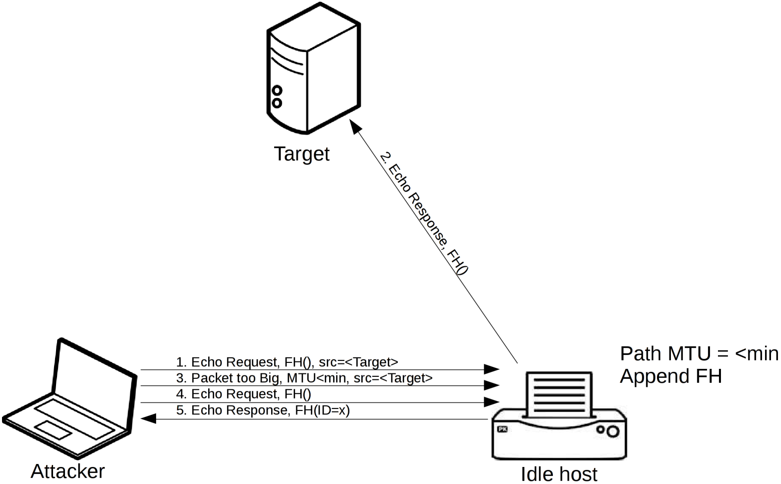
The TCP Idle Scan in IPv6



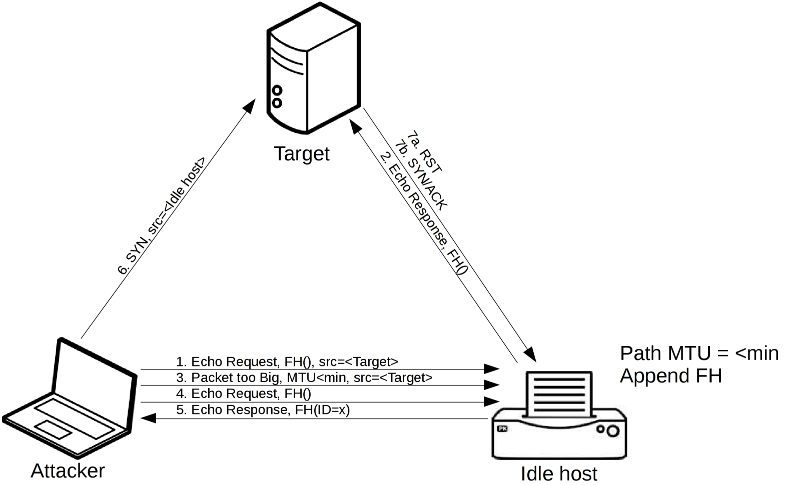
The TCP Idle Scan in IPv6


Target
2. Echo Response, FH()
1. Echo Request, FH(), src=<Target>
3. Packet too Big, MTU<min, src=<Target>
Path MTU = <min
Append FH
Attacker
Idle host

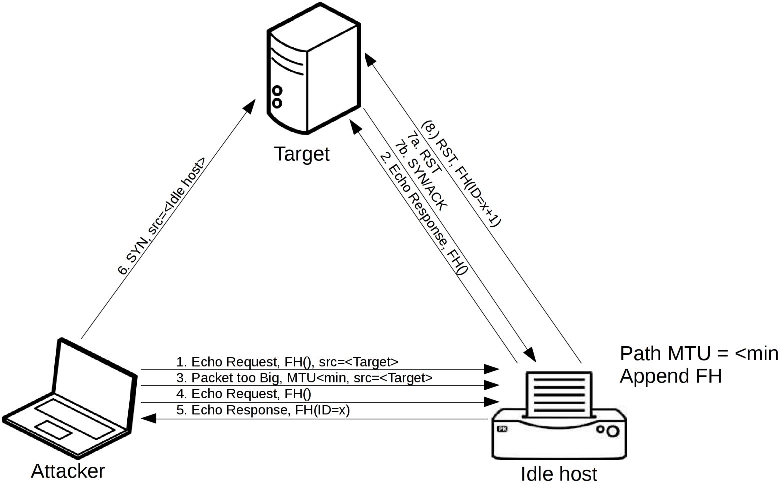
The TCP Idle Scan in IPv6


Target
2. Echo Response, FH()
Path MTU = <min
Append FH
1. Echo Request, FH(), src=<Target>
3. Packet too Big, MTU<min, src=<Target>
4. Echo Request, FH()
5. Echo Response, FH(ID=x)
Attacker
Idle host

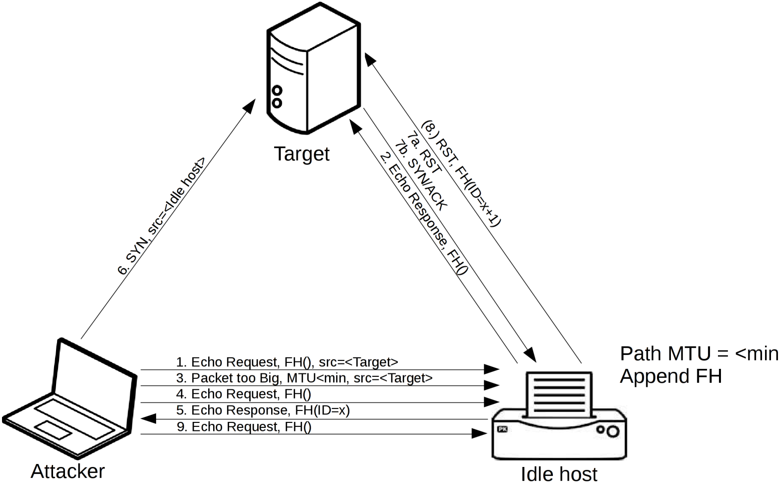
The TCP Idle Scan in IPv6



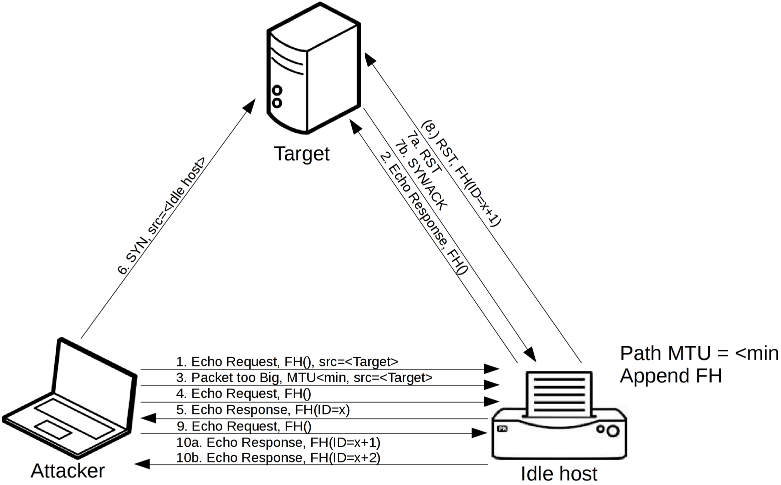
The TCP Idle Scan in IPv6



The TCP Idle Scan in IPv6



The TCP Idle Scan in IPv6



Requirements for Idle host in IPv4

1. Predictable, global assignment of Identification value
2. Remain idle

Requirements for Idle host in IPv6

1. Predictable, global assignment of Identification value ✓

Requirements for Idle host in IPv6

1. Predictable, global assignment of Identification value ✓
2. ~~Remain idle~~
   Do not send fragmented packets

Requirements for Idle host in IPv6

Picture: http://2.bp.blogspot.com/_0Iq8TLRb-Tk/THE2E1s_wkI/AAAAAAAAABQ/KCx-BkbezPs/s1600/huge+wave.jpg

Behavior of different systems

#	System	Assignment of Identification
1	Android 4.1 (Linux 3.0.15)	Per host, incremental (1)
2	FreeBSD 7.4	Random
3	FreeBSD 9.1	Random
4	iOS 6.1.2	Random
5	Linux 2.6.32	Per host, incremental (2)
6	Linux 3.2	Per host, incremental (1)
7	Linux 3.8	Per host, incremental
8	OpenBSD 4.6	Random
9	OpenBSD 5.2	Random
10	OS X 10.6.7	Global, incremental (3)
11	OS X 10.8.3	Random
12	Solaris 11	Per host, incremental

(1) Hosts calculates wrong TCP checksum for routes with PMTU <1280
(2) PMTU <1280 results in DoS
(3) Does not accept PMTU <1280

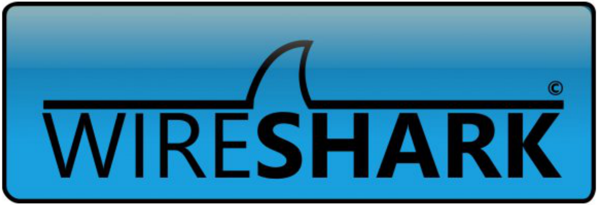
Behavior of different systems

#	System	Assignment of Identification
1	Android 4.1 (Linux 3.0.15)	Per host, incremental (1)
2	FreeBSD 7.4	Random
3	FreeBSD 9.1	Random
4	iOS 6.1.2	Random
5	Linux 2.6.32	Per host, incremental (2)
6	Linux 3.2	Per host, incremental (1)
7	Linux 3.8	Per host, incremental
8	OpenBSD 4.6	Random
9	OpenBSD 5.2	Random
10	OS X 10.6.7	Global, incremental (3)
11	OS X 10.8.3	Random
12	Solaris 11	Per host, incremental
13	Windows Server 2003 R2 Standard 64bit, SP2	Global, incremental
14	Windows Server 2008 Standard 32bit, SP1	Global, incremental
15	Windows Server 2008 R2 Standard 64bit, SP1	Global, incremental by 2
16	Windows Server 2012 Standard 64bit	Global, incremental by 2
17	Windows XP Professional 32bit, SP3	Global, incremental
18	Windows Vista Business 64bit, SP1	Global, incremental
19	Windows 7 Home Premium 32bit, SP1	Global, incremental by 2
20	Windows 7 Ultimate 32bit, SP1	Global, incremental by 2

(1) Hosts calculates wrong TCP checksum for routes with PMTU <1280
(2) PMTU <1280 results in DoS
(3) Does not accept PMTU <1280

Identification value of Windows 8

- Also predictable in Windows 8?



Behavior of different systems

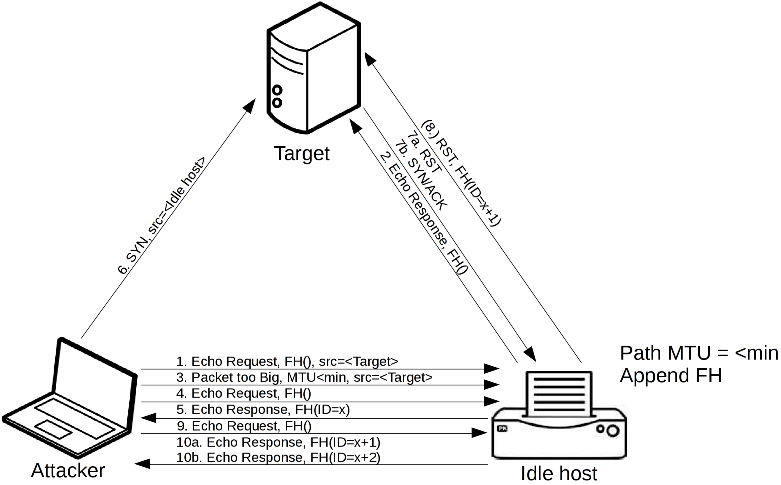
#	System	Assignment of Identification
1	Android 4.1 (Linux 3.0.15)	Per host, incremental (1)
2	FreeBSD 7.4	Random
3	FreeBSD 9.1	Random
4	iOS 6.1.2	Random
5	Linux 2.6.32	Per host, incremental (2)
6	Linux 3.2	Per host, incremental (1)
7	Linux 3.8	Per host, incremental
8	OpenBSD 4.6	Random
9	OpenBSD 5.2	Random
10	OS X 10.6.7	Global, incremental (3)
11	OS X 10.8.3	Random
12	Solaris 11	Per host, incremental
13	Windows Server 2003 R2 Standard 64bit, SP2	Global, incremental
14	Windows Server 2008 Standard 32bit, SP1	Global, incremental
15	Windows Server 2008 R2 Standard 64bit, SP1	Global, incremental by 2
16	Windows Server 2012 Standard 64bit	Global, incremental by 2
17	Windows XP Professional 32bit, SP3	Global, incremental
18	Windows Vista Business 64bit, SP1	Global, incremental
19	Windows 7 Home Premium 32bit, SP1	Global, incremental by 2
20	Windows 7 Ultimate 32bit, SP1	Global, incremental by 2
21	Windows 8 Enterprise 32 bit	Global, incremental by 2

(1) Hosts calculates wrong TCP checksum for routes with PMTU <1280
(2) PMTU <1280 results in DoS
(3) Does not accept PMTU <1280

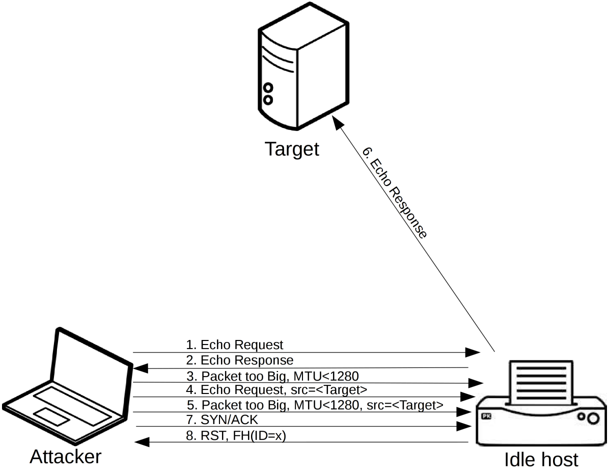
Defense Mechanisms

- Prevent IP-Spoofing
  (Reverse Path Forwarding, Network Ingress Filtering, ...)
- Stateful firewalls
- Random assignment of Identification value

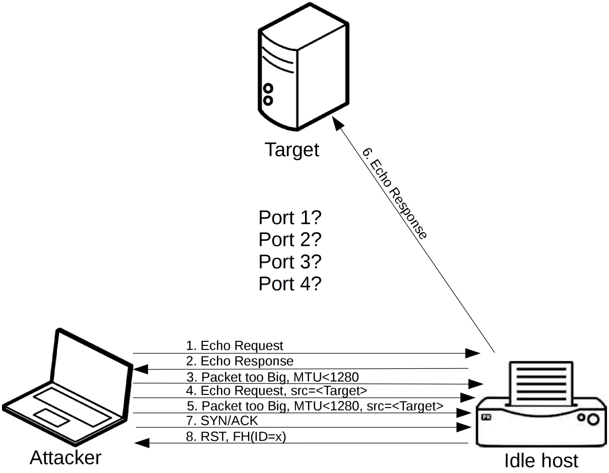
Defense Mechanisms



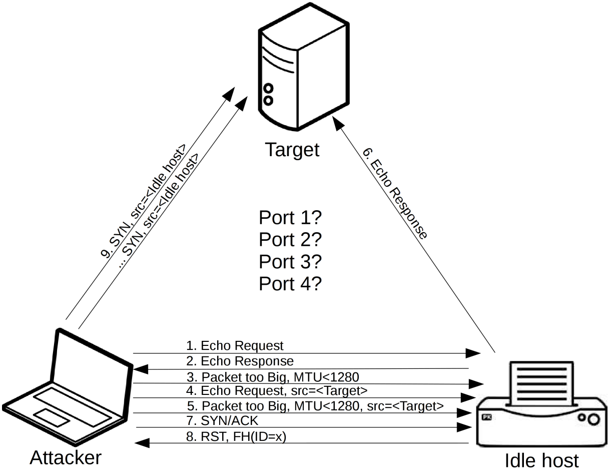
TCP Idle Scan in IPv6 with Nmap


Target
6. Echo Response
1. Echo Request
2. Echo Response
3. Packet too Big, MTU<1280
4. Echo Request, src=<Target>
5. Packet too Big, MTU<1280, src=<Target>
7. SYN/ACK
8. RST, FH(ID=x)
Attacker
Idle host

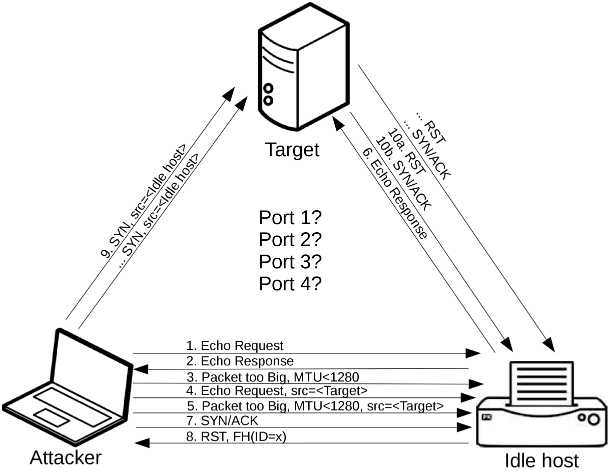
TCP Idle Scan in IPv6 with Nmap



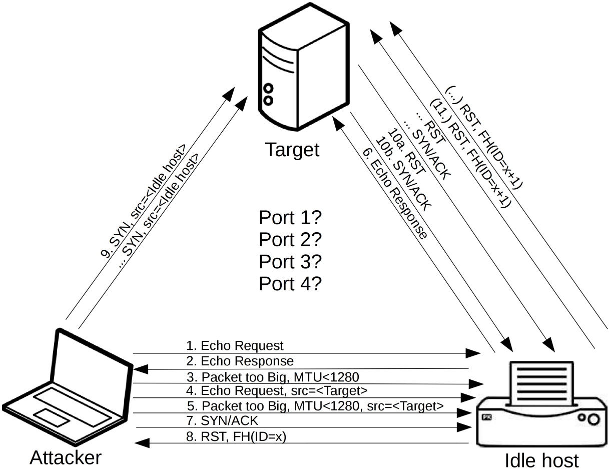
TCP Idle Scan in IPv6 with Nmap



TCP Idle Scan in IPv6 with Nmap



TCP Idle Scan in IPv6 with Nmap



Target

Port 1?
Port 2?
Port 3?
Port 4?

9. SYN, src=<Idle host>
... SYN, src=<Idle host>

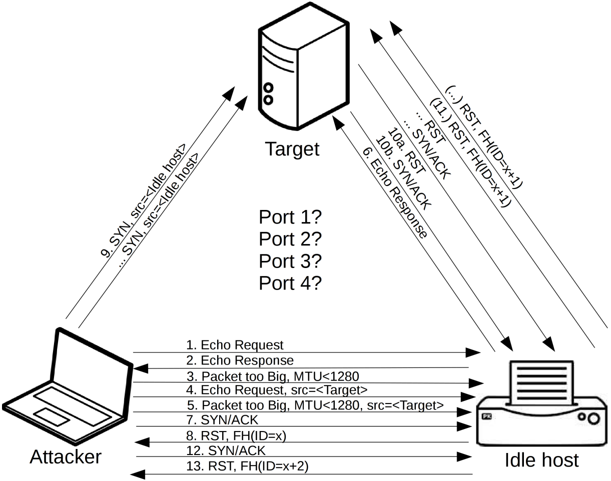
(...) RST, FH(ID=x+1)
(11.) RST, FH(ID=x+1)
... RST
... SYN/ACK
10a. RST
10b. SYN/ACK
6. Echo Response

1. Echo Request
2. Echo Response
3. Packet too Big, MTU<1280
4. Echo Request, src=<Target>
5. Packet too Big, MTU<1280, src=<Target>
7. SYN/ACK
8. RST, FH(ID=x)

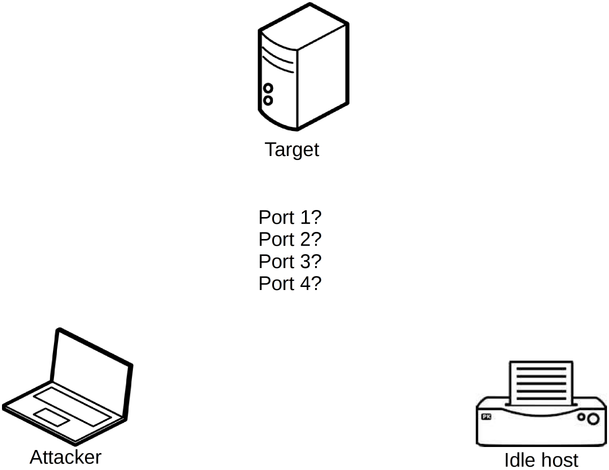
Attacker

Idle host

TCP Idle Scan in IPv6 with Nmap

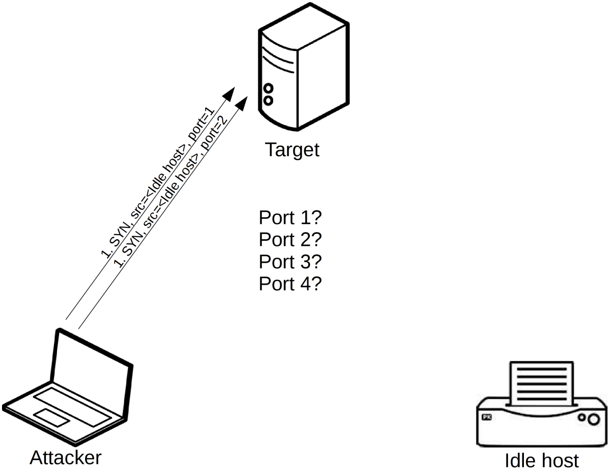


Find the open port



Port 1?
Port 2?
Port 3?
Port 4?

Find the open port



Target

1. SYN, src=<Idle host>, port=1

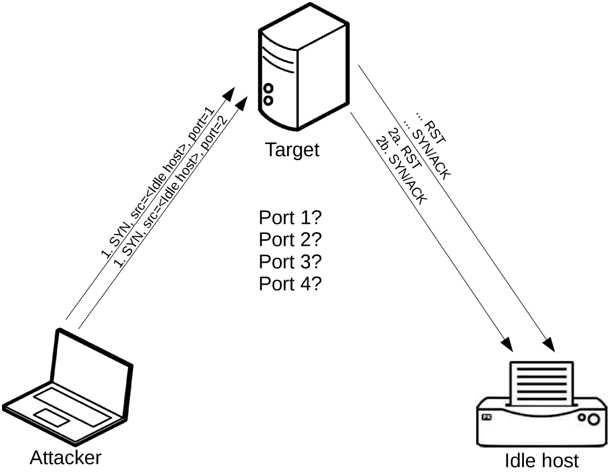
1. SYN, src=<Idle host>, port=2

Port 1?
Port 2?
Port 3?
Port 4?

Attacker

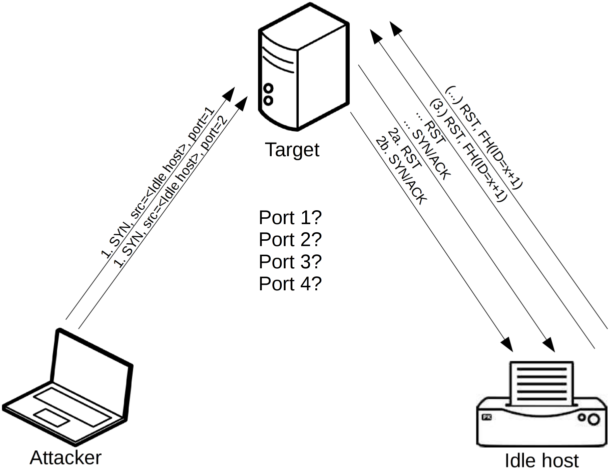
Idle host

Find the open port


Target
1. SYN, src=<Idle host>, port=1
1. SYN, src=<Idle host>, port=2
... RST
... SYN/ACK
2a. RST
2b. SYN/ACK
Attacker
Idle host

Port 1?
Port 2?
Port 3?
Port 4?

Find the open port



Target

Attacker

Idle host

1. SYN, src=<Idle host>, port=1

1. SYN, src=<Idle host>, port=2

2a. RST

2b. SYN/ACK

... RST

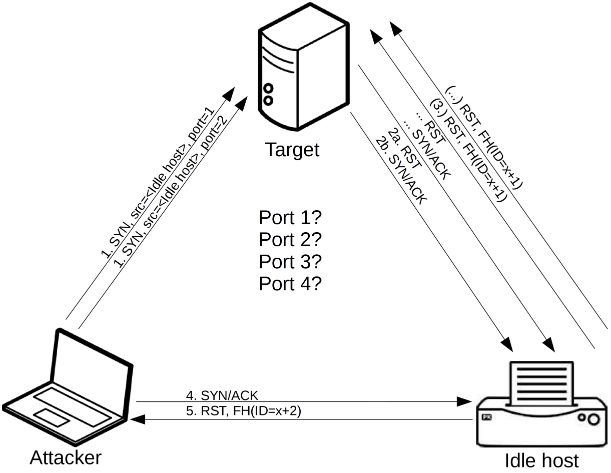
... SYN/ACK

(3.) RST, FH(ID=x+1)

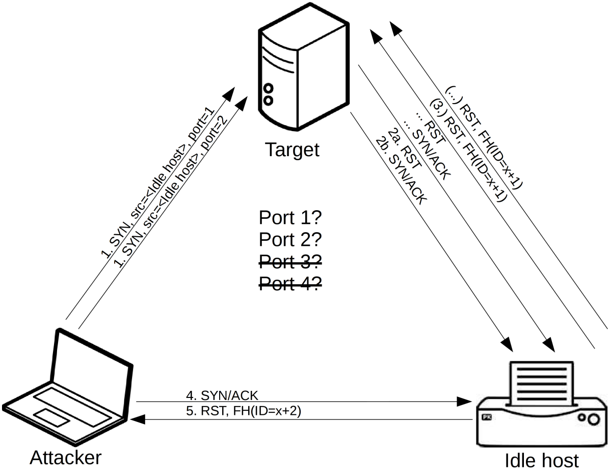
(...) RST, FH(ID=x+1)

Port 1?
Port 2?
Port 3?
Port 4?

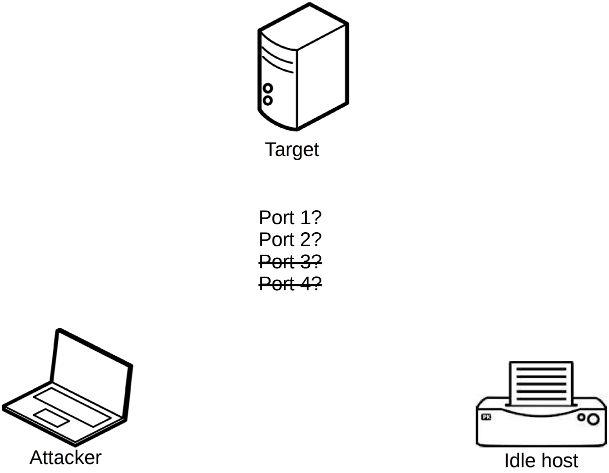
Find the open port



Find the open port



Find the open port



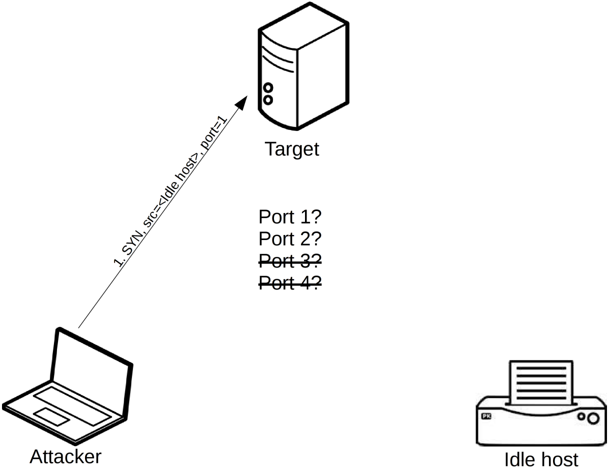
Target

Port 1?
Port 2?
~~Port 3?~~
~~Port 4?~~

Attacker

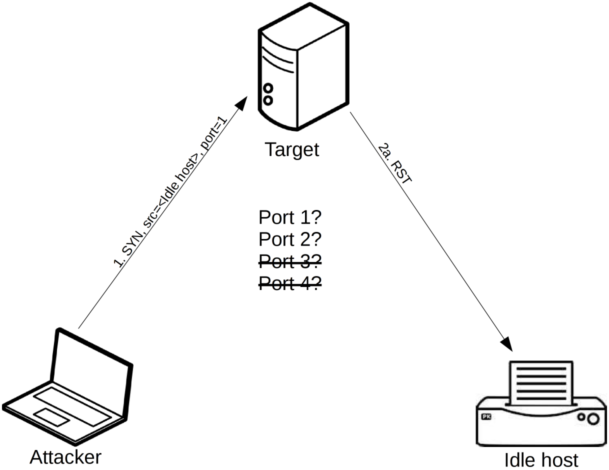
Idle host

Find the open port

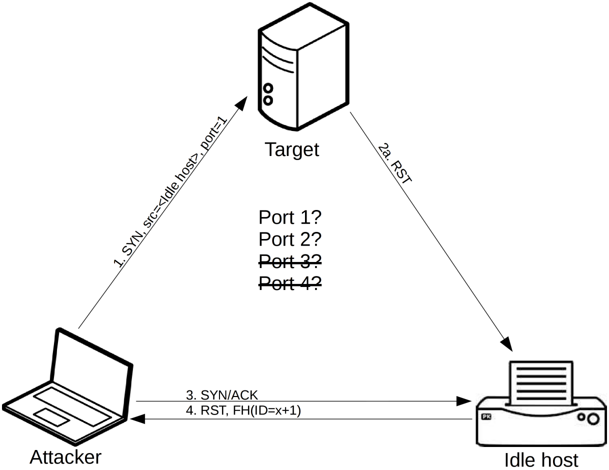


Port 1?
Port 2?
~~Port 3?~~
~~Port 4?~~

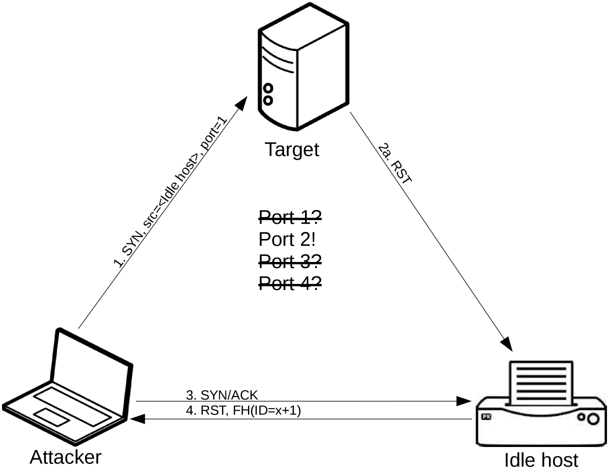
Find the open port



Find the open port



Find the open port



Implementation

- Nmap implementation of TCP Idle Scan in IPv6
- 8.13s to scan 1000 ports in IPv6

Implementations

- Nmap implementation of TCP Idle Scan in IPv6
- 8.13s to scan 1000 ports in IPv6
  8.06s to scan 1000 ports in IPv4

Implementations

- Nmap implementation of TCP Idle Scan in IPv6
- 8.13s to scan 1000 ports in IPv6
  8.06s to scan 1000 ports in IPv4
  → loss of less than 1% performance while having less requirements

Implementations

- Nmap implementation of TCP Idle Scan in IPv6
- 8.13s to scan 1000 ports in IPv6
  8.06s to scan 1000 ports in IPv4
  → loss of less than 1% performance while having less requirements
- Soon to be in the official release

Conclusion

- Lessons learned: None?
- Danger of predictable IDs shown in 1985 (TCP)

Conclusion

- Lessons learned: None?
- Danger of predictable IDs shown in 1985 (TCP)
- Proven with the TCP Idle Scan in 1998 (IPv4)

Conclusion

- Lessons learned: None?
- Danger of predictable IDs shown in 1985 (TCP)
- Proven with the TCP Idle Scan in 1998 (IPv4)
- Feasible again in IPv6 in 2013!

Conclusion

DO NOT USE PREDICTABLE IDs,

DAMN IT!

Questions

Picture: http://www.hdallwallpapers.com/wp-content/uploads/2013/08/despicable_me_2_minion-1600x1200.jpg