IPv6 Extension Headers - New Features, and New Attack Vectors

Antonios Atlasis
antonios.atlasis@cscss.org

Troopers 13 – IPv6 Security Summit 2013

Bio

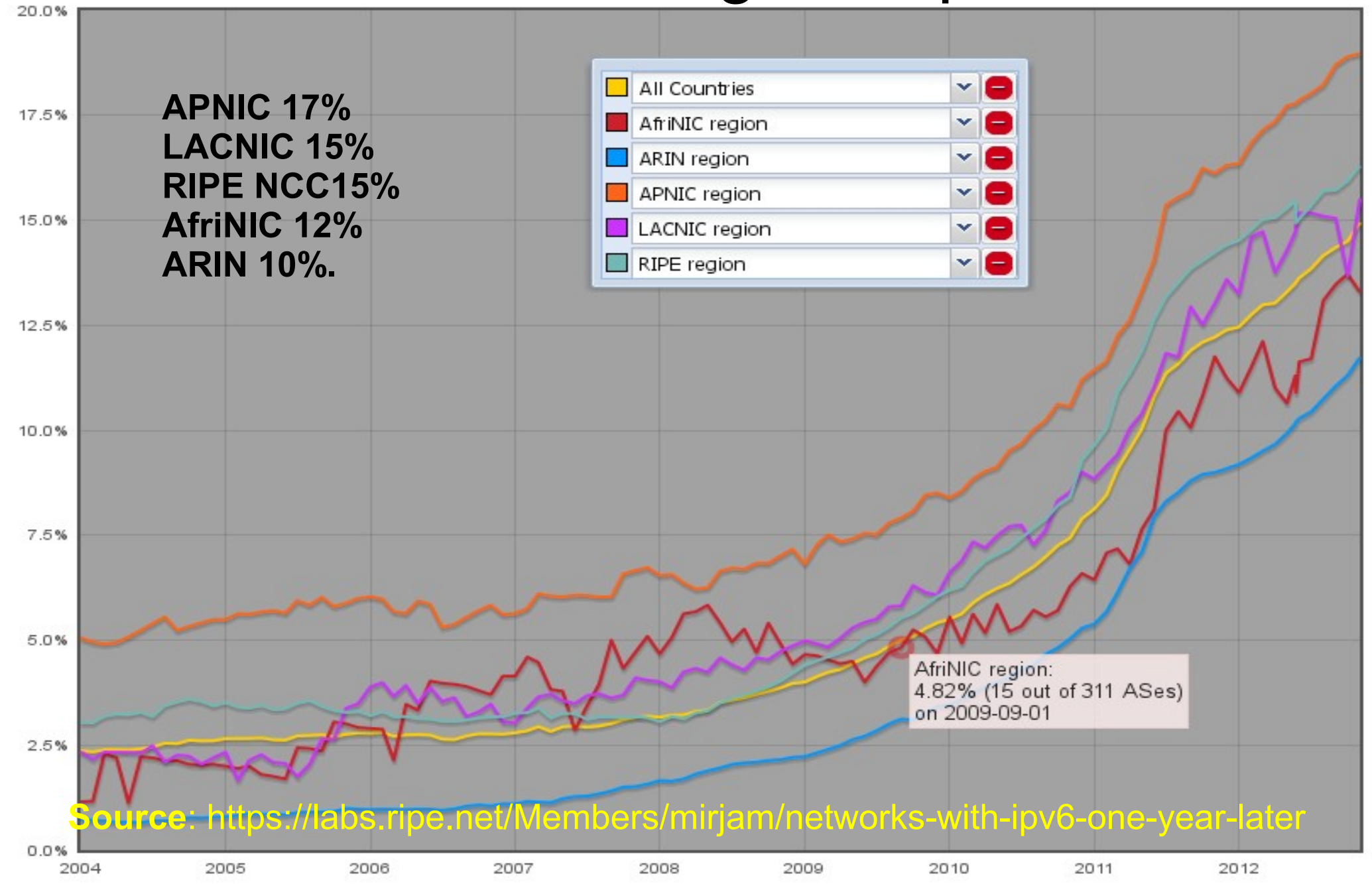
- Independent IT Security analyst/researcher.
- **MPhil** Univ. of Cambridge, **PhD** NTUA, etc.
- Over 20 years of diverse Information Technology experience.
- Instructor and software developer, etc.
- More than 25 scientific and technical publications in various IT fields.
- Last one and a half year studying IPv6 and especially potential security implications due to the misuse of IPv6 Extension Headers.
- E-mail: antonios.atlasis@cscss.org

  antonios.atlasis@gmail.com

Agenda

- Introduction
- The IPv6 Extension Headers
- Abusing IPv6 Extension Headers
- Tested scenarios – Results
- Security impacts of abusing IPv6 Extension Headers - Demos
- Proposed countermeasures
- Conclusions

Percentage of Autonomous Sytems announcing IPv6 prefixes



Source: https://labs.ripe.net/Members/mirjam/networks-with-ipv6-one-year-later

IPv6 @ the Gates

- 6th June of 2012, the IPv6 world launch day.
- "***IPv6-ready***" products, such as Operating Systems, Networking Devices, Security Devices, etc.
- IPv6 is offered by several ISPs worldwide, even from smaller countries.
- The time for IPv6 has finally come.

What does a new protocol introduce?

- New features, new capabilities, …
- but also new potential vulnerabilities and hence, new attack vectors (hackers/crackers joy).
- IPv6 is around for many years, but it has not been tested operationally yet, at least not extensively.

Security Implications of Attacking a Network Protocol?

- A Layer-7 protocol:

  Only this protocol is affected.

- A Layer-3 protocol:

  ALL the above protocols are affected (can be disastrous).

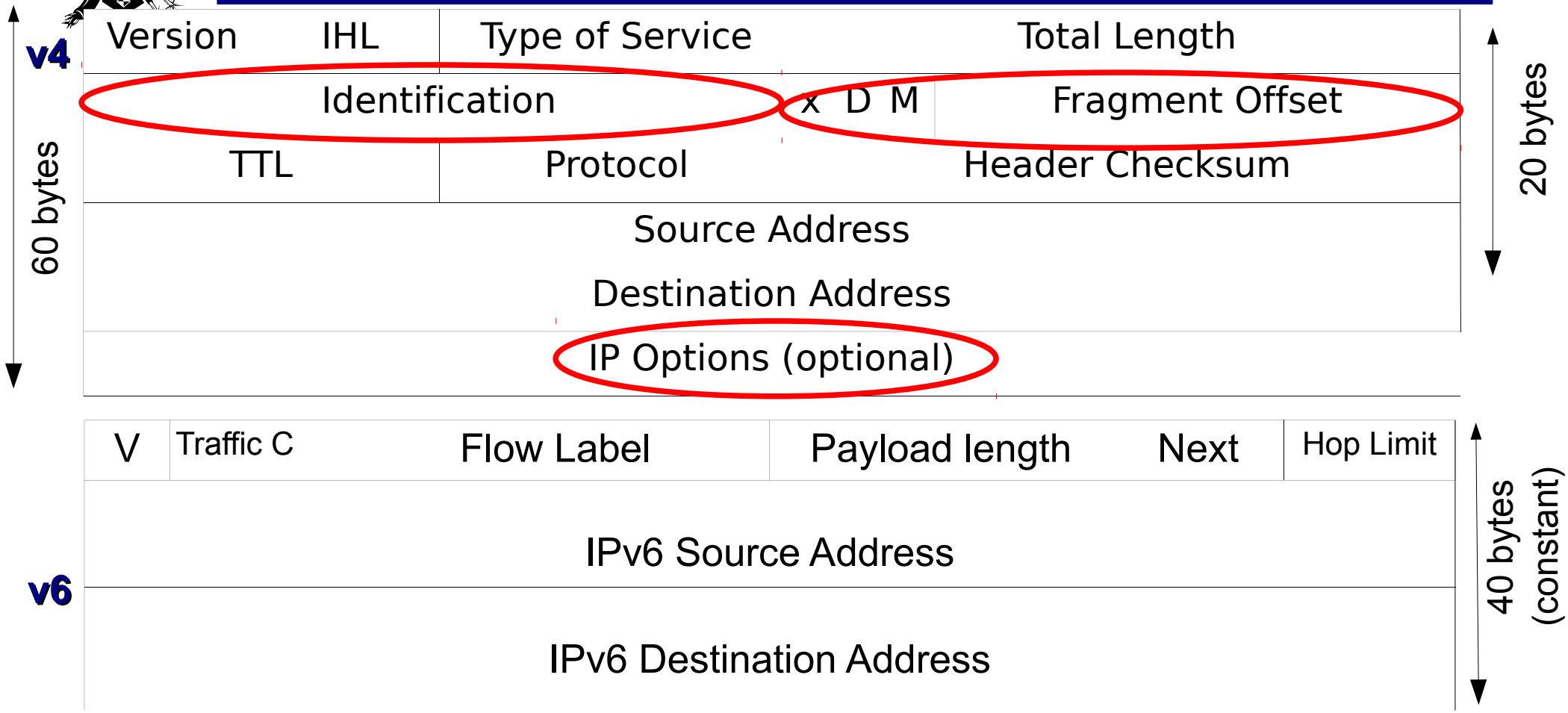
IPv6 Potential Security Issues

- Two categories:
  - Issues known from the IPv4 era, solved in IPv4 but re-appear in IPv6. Examples: Layer-4 Fragmentation overlapping, predicted fragmentation ID values, etc.
  - Issues new to IPv6 introduced due to its new features.

IPv6 New Features

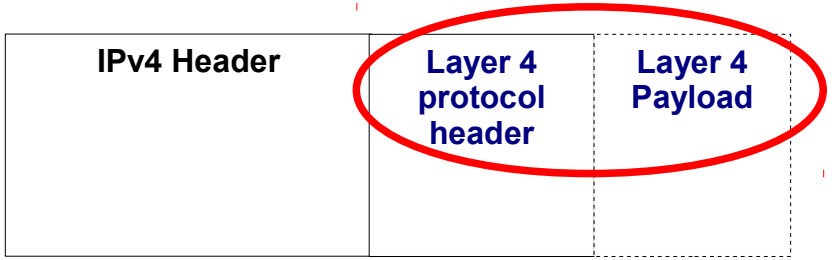
- It is not just the huge address space.
- One of the most significant changes: The introduction of the **IPv6 Extension Headers**.
- Let's remember how they SHOULD be used.

The IPv4 vs the IPv6 Header

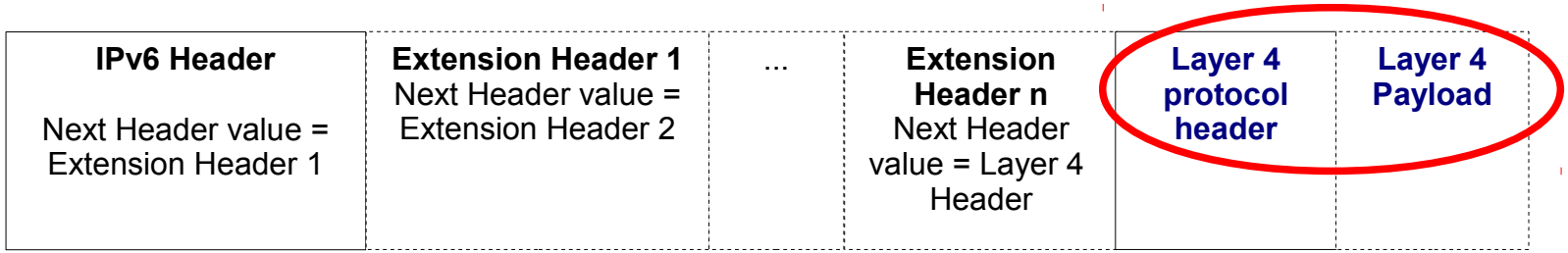


IPv6 Extension headers have been introduced to support any extra functionality, if required.

An IPv6 vs an IPv4 Datagram



IPv4 datagram



IPv6 datagram

The IPv6 Extension Headers (RFC 2460)

- Hop-by-Hop Options [RFC2460]
- Routing [RFC2460]
- Fragment [RFC2460]
- Destination Options [RFC2460]
- Authentication [RFC4302]
- Encapsulating Security Payload [RFC4303]
- MIPv6, [RFC6275] (Mobility Support in IPv6)
- HIP, [RFC5201] (Host Identity Protocol)
- shim6, [RFC5533] (Level 3 Multihoming Shim Protocol for IPv6)
- **All (but the Destination Options header) SHOULD occur at most once.**
- **How a device should react if NOT?**

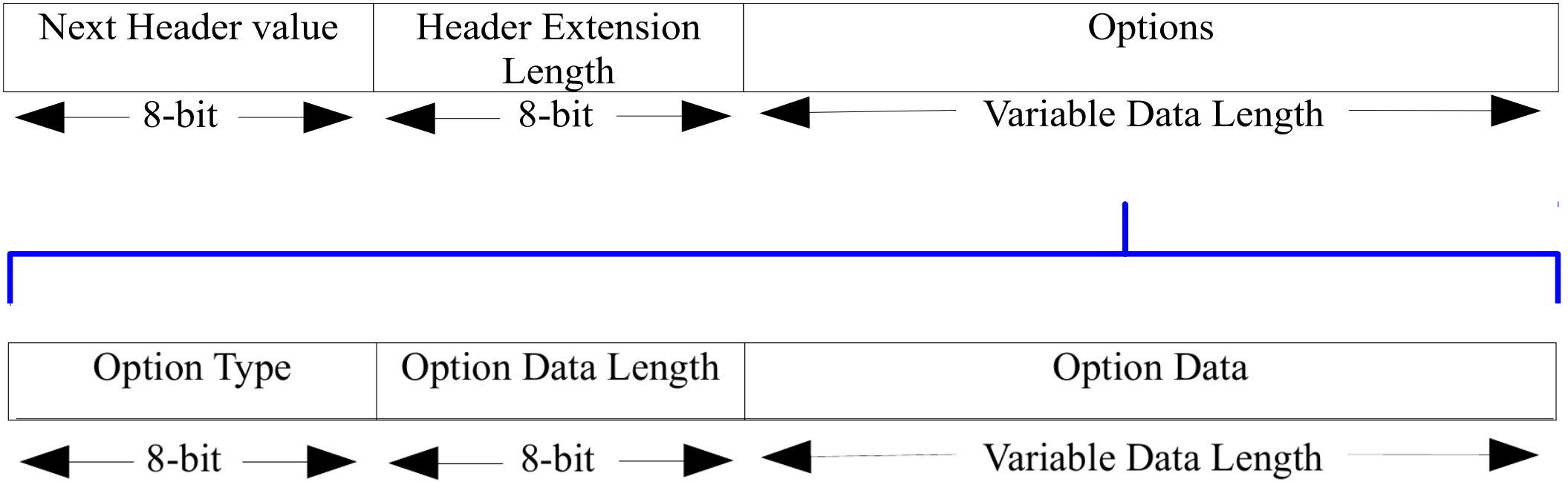
Recommended IPv6 Extension Headers Order

- IPv6 header
- Hop-by-Hop Options header
- Destination Options header
- Routing header
- Fragment header
- Authentication header
- Encapsulating Security Payload header
- Destination Options header (for options to be processed only by the final destination of the packet.)
- Upper-layer header

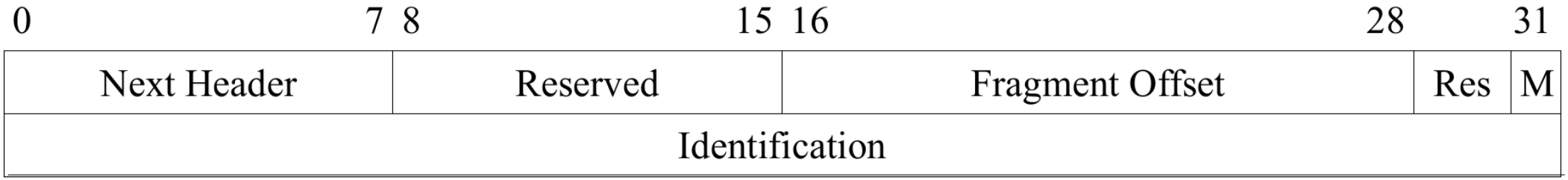
Abuse of IPv6 Extension Headers

- Two Extension Headers will be tested here:
  - the Destination Options Header
  - and the Fragment Extension header
- In some of the tested scenarios other IPv6 Extension Headers can also be used.

The Destination Options Header



The IPv6 Fragment Header



- The M bit, the Identification number and the Offset have moved here from the main header.
- The DF bit has been totally removed.

Abusing IPv6 Extension Headers

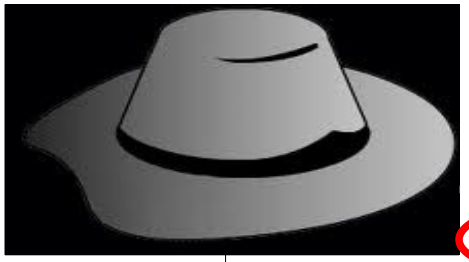
- RFCs describe the way that IPv6 Extension Headers has to or should be used.
- In either case, this does not mean that the vendors make RFC compliant products.
- RFCs do not specify how the OS should react in a different case → increase the ambiguity → if exploited properly, can lead to various security flaws.

Creating Tested Scenarios

- Based on the RFC definitions, several what-if scenarios can be created.
  - What-if the order is different, what-if there are more headers of some types than recommended, what-if we combine several situations, etc.
- Based on the findings, we 'll try to “exploit” them for security reasons.

The Lab Environment

attacker



Scapy scripts

OpenBSD 5.1/5.2

fed0::5/64
fed0::52/64

Centos 6.3

fed0::63/64

FreeBSD 9/9.1

fed0::9/64
fed0::91/64

Windows 7

fed0::7/64

Windows 8

fed0::8/64

Ubuntu

10.04

fed0::10/64

Ubuntu

12.04

fed0::12/64

Windows Server 2008

fed0::2008/64

Used Protocol during Tests

- As an upper-layer protocol, the ICMPv6 was used (Echo Request type):
  - It is the simplest protocol that can invoke a response.
  - It also echoes back the payload of the Echo Request packet
- Hence, using unique payload per packet, the fragmentation reassembly policy of the target can be easily identified.

Our Attacking Tool

- **Scapy**
  - A powerful interactive packet manipulation program.
  - http://www.secdev.org/projects/scapy/
  - Requires Python 2.5 or greater.
  - Supports (among else) IPv6 headers in its latest (dev) version.

IPv6 functions in Scapy

- **IPv6**: IPv6 header
- **IPv6ExtHdrDestOpt** : IPv6 Destination Options Header
- **IPv6ExtHdrFragment** : IPv6 Fragmentation header
- **IPv6ExtHdrHopByHop** : IPv6 Hop-by-Hop Options Header
- **IPv6ExtHdrRouting** : IPv6 Option Header Routing
- Several ICMPv6 types (we will use the **ICMPv6EchoRequest**).

Creating an IPv6 Header

```
>>> IPv6().show()
###[ IPv6 ]###
   version= 6
   tc= 0
   fl= 0
   plen= None
   nh= No Next Header
   hlim= 64
   src= ::1
   dst= ::1
```

nh (next header) should be 44 if the next header is a Fragment Extension header.

Creating an IPv6 Fragment Extension Header

```
>>> IPv6ExtHdrFragment().show()
###[ IPv6 Extension Header - Fragmentation header ]###
  nh= No Next Header
  res1= 0
  offset= 0
  res2= 0
  m= 0
  id= None
```

m: More fragments to follow bit.
nh (next header): Should be 58 if ICMPv6 Echo Request is the next header.

ICMPv6 Echo Request Crafting

```
>>> ICMPv6EchoRequest().show()
###[ ICMPv6 Echo Request ]###
   type= Echo Request
   code= 0
   cksum= None
   id= 0x0
   seq= 0x0
   data= ''
```

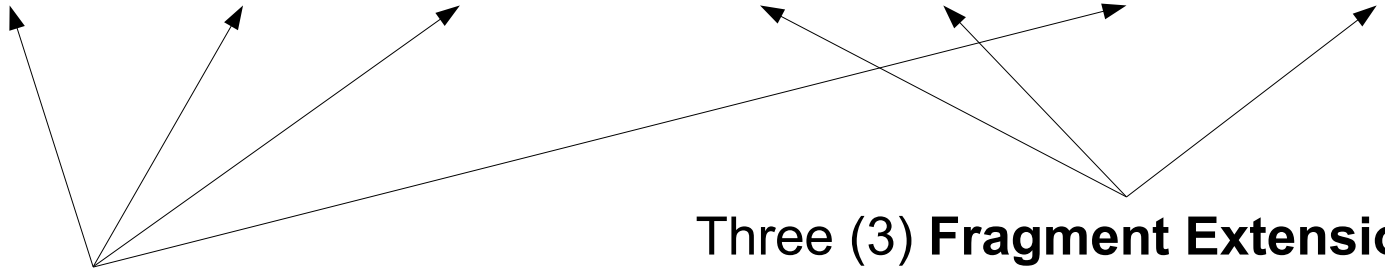
data: The ICMPv6 payload.
Special attention to checksum (**csum**) computation.

Basic Groups of Tested Scenarios

- More than one occurrences of various extension headers in atomic fragments.
- Nested fragments (that is, ...fragmented fragments).
- Sending the upper-layer protocol header at a fragment other than the 1st one.
- Creating overlapping extension headers (3 cases will be examined).
- Transfer of arbitrary data at the IP level (fragmented or not).
- IPv6 in IPv6 in IPv6, ... (and also ...fragmented).

1. Multiple Occurrences of Various Extension Headers in an Atomic Fragment

IPv6 Header	Destination Options Header	Destination Options Header	Destination Options Header	Fragment Header	Fragment Header	Destination Options Header	Fragment Header	ICMPv6 EchoRequest Header



Three (3) **Fragment Extension Headers**

Four (4) **Destination Options Headers**

1. Multiple Occurrences of Various Extension Headers in an Atomic Fragment

*send(IPv6(src=sip, dst=dip) *

/IPv6ExtHdrDestOpt() \

/IPv6ExtHdrDestOpt() \

/IPv6ExtHdrDestOpt() \

/IPv6ExtHdrFragment (offset=0, m=0) \

/IPv6ExtHdrFragment(offset=0, m=0) \

/IPv6ExtHdrDestOpt() \

/IPv6ExtHdrFragment(offset=0, m=0) \

/ICMPv6EchoRequest())

1. Multiple Occurrences of Various Extension Headers in an Atomic Fragment

- Such a packet SHOULD NOT exist, but how the OS should react?
- Demo 1
- Results:
  - OpenBSD was the only one that does not accept such a malformed packet.
  - Similar results even if only one type of an Extension Header is repeated more than once.

2. Nested Fragments

IPv6 Header	Fragment Header #1	Fragment Header #2	
	Outer fragment header	Iner fragment header	
IPv6 Header	Fragment Header #1	Fragment Header #2	Packet 2
...	Outer fragment header	Iner fragment header	
IPv6 Header	Fragment Header #1	Fragment Header #2	Packet n

2. Nested Fragments

*ipv6_1=IPv6(src=sip, dst=dip, plen=8*2)*

*frag2=IPv6ExtHdrFragment(offset=0, m=0, id=**myid2**, **nh=44**)*

for i in range(0, no_of_fragments):

*frag1=IPv6ExtHdrFragment(offset=i, m=1, **id=myid**, **nh=44**)*

packet=ipv6_1/frag1/frag2

send(packet)

*frag1=IPv6ExtHdrFragment(offset=no_of_fragments, m=1, **id=myid**, **nh=44**)*

*frag2=IPv6ExtHdrFragment(offset=0, m=0, **id=myid2**, **nh=58**)*

packet=ipv6_1/frag1/frag2

send(packet)

ipv6_1=IPv6(src=sip, dst=dip, plen=8(length+1))*

*frag1=IPv6ExtHdrFragment(offset=no_of_fragments+1, m=0, **id=myid**, **nh=44**)*

packet=ipv6_1/frag1/icmpv6

send(packet)

2. Nested Fragments

- There is no reason for a legitimate user to create nested fragments.
- Demo 2

2. Nested Fragments

- Results:
  - The Windows and the Ubuntu systems respond back with an ICMPv6 Echo Reply message, meaning that these accept these malformed messaged.
  - Centos 6.3, FreeBSD and OpenBSD don't.
  - NOTICE: Different behaviour between Centos and Ubuntu 10.04, although they actually use the same kernel.

3. Upper-layer Protocol Header at a Fragment other than the 1st Fragment

IPv6 Header	Fragment Header	Destination Options Header	Packet 1
IPv6 Header	Fragment Header	Destination Options Header	Packet 2
IPv6 Header	Fragment Header	ICMPv6 Plus payload	Packet 3

3. Upper-layer Protocol Header at a Fragment other than the 1st Fragment

```
packet1 = IPv6(src=sip, dst=dip) \
      /IPv6ExtHdrFragment(offset=0, m=1) \
      /IPv6ExtHdrDestOpt(nh=60)
packet2 = IPv6(src=sip, dst=dip) \
      /IPv6ExtHdrFragment(offset=1, m=1) \
      /IPv6ExtHdrDestOpt(nh=58)
packet3 = IPv6(src=sip, dst=dip) \
      /IPv6ExtHdrFragment(offset=2, m=0, nh=58) \
      /ICMPv6EchoRequest(cksum=csum, data=payload1)
send(packet1)
send(packet2)
send(packet3)
```

3. Upper-layer Protocol Header at a Fragment other than the 1st Fragment

- Demo 3
- Results:
  - OpenBSD, the Ubuntu and the Windows hosts accept the datagrams, although the checksum appears to be incorrect.
  - FreeBSD 9/9.1 and Centos 6.3 don't.

4.Mixing Extension Headers and Sending the Upper-Layer Protocol Header at a Fragment other than the 1st

- A combination of the 1st (mixing multiple extension headers) and the 3rd (sending the upper layer header at a fragment other than the 1st) scenarios.

4.Mixing Extension Headers and Sending the Upper-Layer Protocol Header at a Fragment other than the 1st

packet1 *= IPv6(src=sip, dst=dip)* \
/IPv6ExtHdrFragment(offset=0, m=1) \
/IPv6ExtHdrDestOpt(nh=60) **** \
/IPv6ExtHdrDestOpt(nh=60) \
/IPv6ExtHdrDestOpt(nh=60) \
/IPv6ExtHdrDestOpt(nh=60) \
/IPv6ExtHdrDestOpt(nh=58)

Five (5) Destination Option headers!

packet2 *= IPv6(src=sip, dst=dip)* \
/IPv6ExtHdrFragment(**offset=5**, m=0, nh=58) \
/ICMPv6EchoRequest*(cksum=csum, data=payload1)*

send(packet1)

send(packet2)

Layer 4 header at the 2nd fragment

4.Mixing Extension Headers and Sending the Upper-Layer Protocol Header at a Fragment other than the 1st

- Demo 4.
- Results:
  - Only FreeBSD 9/9.1 do not accept such packets.
  - All the others (included OpenBSD that discards such combinations in atomic fragments and Centos 6.3 that discarded before) DO accept them (although the checksum appears to be incorrect).
  - Remark: By combining two methods, both Centos 6.3 and OpenBSD 5.2 accept the malformed packets.

Creating Overlapping Extension headers

- This is a layer-3 overlapping, not an overlapping known from IPv4.
- Case 1:

  The 3rd fragment overlaps the 2nd.
- Case 2:

  The 3rd fragment overlaps the 1st.

5. Creating Overlapping Extension headers: Case 1

*packet1 = IPv6(src=sip, dst=dip) *

*/IPv6ExtHdrFragment(offset=0, m=1) *

/IPv6ExtHdrDestOpt(nh=58)

*packet2 = IPv6(src=sip, dst=dip) *

*/IPv6ExtHdrFragment(**offset=1**, m=1, nh=58) *

/IPv6ExtHdrDestOpt(nh=58)

***packet3** = IPv6(src=sip, dst=dip) *

*/IPv6ExtHdrFragment(**offset=1, m=0**, nh=58) *

/ICMPv6EchoRequest(cksum=csum, data=payload1)

send(packet1)

send(packet2)

send(packet3)

6. Creating Overlapping Extension headers: Case 2

*packet1 = IPv6(src=sip, dst=dip) *

*/IPv6ExtHdrFragment(**offset=0**, m=1) *

/IPv6ExtHdrDestOpt(nh=58)

*packet2 = IPv6(src=sip, dst=dip) *

*/IPv6ExtHdrFragment(offset=1, m=1, nh=58) *

/IPv6ExtHdrDestOpt(nh=58)

***packet3** = IPv6(src=sip, dst=dip) *

*/IPv6ExtHdrFragment(**offset=0, m=0**, nh=58) *

/ICMPv6EchoRequest(cksum=csum, data=payload1)

send(packet1)

send(packet2)

send(packet3)

5. Creating Overlapping Extension headers: Case 1

- Another quick demo (5):
- Results:
  - Centos 6.3 and Ubuntu 10.04 accept the malformed packets (“old” but PATCHED linux kernels).
- Remember: These are many Linux Enterprise systems.

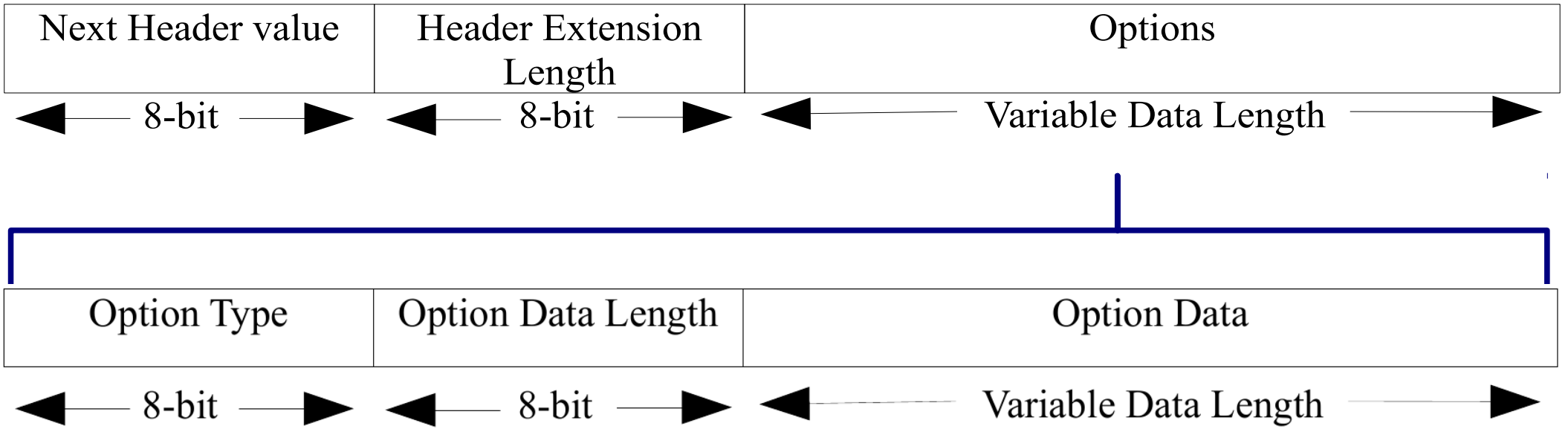
6-7. Creating Overlapping Extension headers: Case 2

- All the Linux systems (Centos 6.3 and Ubuntu) respond back to such malformed packets.
- FreeBSD 9.1 does accept such packets, while FreeBSD 9 don't.
- Similar results when there are only two fragments, with the 2nd one overlapping the 1st.
- So, in this case, FreeBSD 9.1 and Ubuntu 12.04 are added.

8. Transfer of arbitrary data at the IP level

- The IPv6 ***Destination Options Extension*** header and the ***Hop-by-Hop Options*** header carry a variable number of type-length-value (TLV) encoded “options”.

The Destination Options Header



If the two highest-order bits of the “Option Type” are equal to 01, the recipient should discard the packet.

if we put arbitrary data into such a header using this specific Options Type, this data will be transferred even if they do not form a valid packet.

8. Transfer of arbitrary data at the IP level

```
packet = IPv6(src=sip, dst=dip) \
      /IPv6ExtHdrDestOpt(options=PadN(optdata='\101'*120) \
      /PadN(optdata='\102'*150) \
      /PadN(optdata='\103'*15)) \
      /ICMPv6EchoRequest()
send(packet)
```

A's

B's

C's

8. Transfer of arbitrary data at the IP level

- All the tested OS accept such a packet.
- Officially, this is not a bug, since this is what the RFC2460 recommends.
- However, it has its own security impact.

9. Transfer of arbitrary data at the IP level

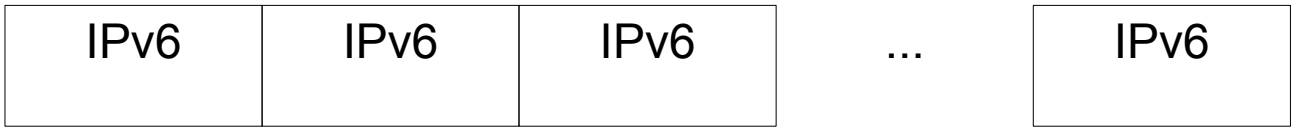
- We can expand the room for arbitrary data, by using several such Extension Headers in a packet, or several fragments.
- OpenBSD (for 8 fragments or less), Windows and Ubuntu accept that.
- Again, different behaviour between Linuces with the same kernel.

What else RFCs say to us?

- RFC 2460: "*If the upper-layer header is another IPv6 header (in the case of IPv6 being tunneled over or encapsulated in IPv6), it may be followed by its own extension headers, which are separately subject to the same ordering recommendations.*"

What if we Tunnel IPv6 in IPv6?

- Is this (officially) allowed?



- How an OS should respond on this?

Simple Code (again)

```
for i in range(1, number_of_headers):
    if i==1:
        packet=IPv6(src=sip2,
        dst=ip)/ICMPv6EchoRequest(id=icmpid,data=p
        ayload)
    else:
        packet=IPv6(src=sip2, dst=ip)/packet
packet=IPv6(src=sip, dst=ip)/packet
```

IPv6 Tunneled in IPv6

- Demo 6
- OK, but in which source (if different in each IPv6 header) does the recipient respond?
- What if we fragment IPv6 tunneled traffic?
- Demo 7.

	Centos 6.3 2.6.32-279.22.1	Ubuntu 10.04.4 2.6.32-45	Ubuntu 12.04.1 3.2.0-37	FreeBSD 9.0p3/9.1	OpenBSD 5.2	Windows XP/7/8/2008
1. Mixing Multiple and Various Extension Headers per datagram in atomic fragments	√	√ *	√	√/√		√
2. Nested fragments		√	√			√
3.Upper-layer Protocol Header at Fragment other than the 1st		√	√		√	√
4. Upper-layer Protocol Header at the 2nd Fragment and Mixing Multiple Extension headers at the 1st	√	√	√		√	√
5. Upper-layer Protocol Header at the Third Fragment with the 3rd fragment overlapping the 2nd	√	√				
6-7. Upper-layer Protocol Header at the Third Fragment with the 3rd fragment overlapping the 1st (or the 2nd overlaps the 1st)	√	√ *	√	/√		
8. Transfer of "large" amount of arbitrary data at the IP level	√	√	√	√/√	√	√
9 Transfer of "large" amount of fragmented arbitrary data at the IP level		√	√		√	√
10. IPv6 tunneled in IPv6	***	Not tested	**	**		√/* */* */* *
11. Fragmented IPv6 tunneled in IPv6.	***	Not tested	**	**		√/* */ /* *

* * ICMPv6 Parameter problem unrecognized Next Header type encountered

* * * Destination unreachable Communication with destination administratively prohibited

* Ubuntu 10.04 LTS responds twice (sends to ICMPv6 Echo Reply messages back to a single ICMPv6 Echo Request message).

Security Impacts of the Misuse of the IPv6 Extension Headers

- OS Fingerprinting (different OS behaviours under different scenarios create detection opportunities).
- Creation of Covert Channels at the IP level.
- Firewall evasion
- Evading Intrusion Detection Systems.
- Remote DoS or code execution?

Covert Channels (before)

- Hiding data - the old ways:
  - At the application layer (e.g. DNS, HTTP, etc.)
    - Easily detectable
  - IPv4 → “Options” Field
    - Very limited space.

Covert Channels (using IPv6)

- Destination Options or Hop-by-hop Extension Header
  - Up to 2048 bytes per IPv6 Dest Opt or Hop-by-hop Extension header.
  - Many headers per packet → big space
  - Not easily detectable (at least yet)
  - Can be encapsulated e.g. in Teredo.
  - We can send legitimate data at the application layer protocol to mislead any detectors.
- **Can your DLP detect this?**

Evading Firewalls

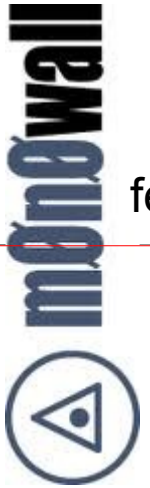
- Remember tunneled traffic accepted by Windows XP?
- Let's see what we can do...

Firewall Evasion Scenario

Legitimate user

fed0::1

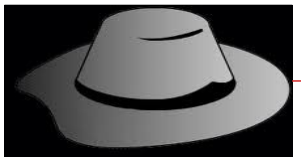
fed0::6



fec0::1

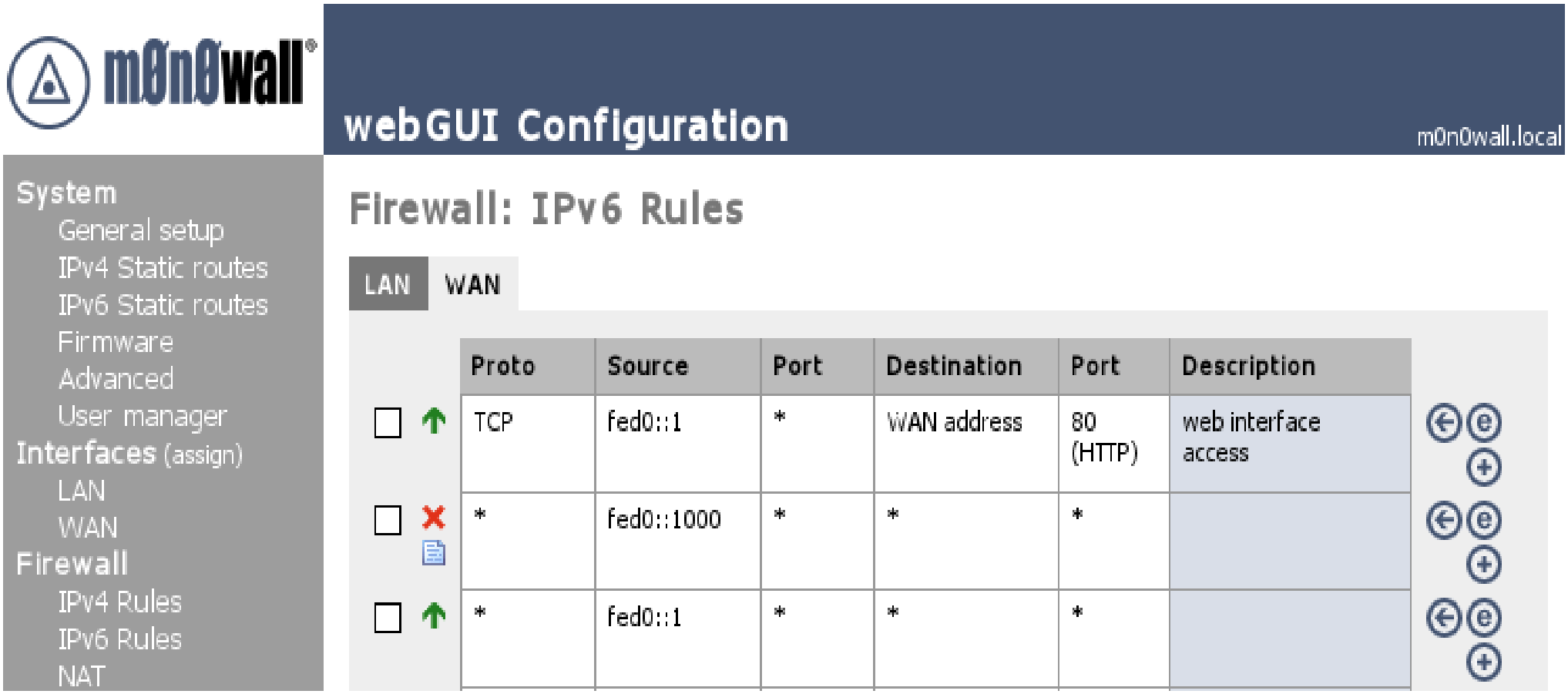
fec0::2002

tcp 135



fed0::1000

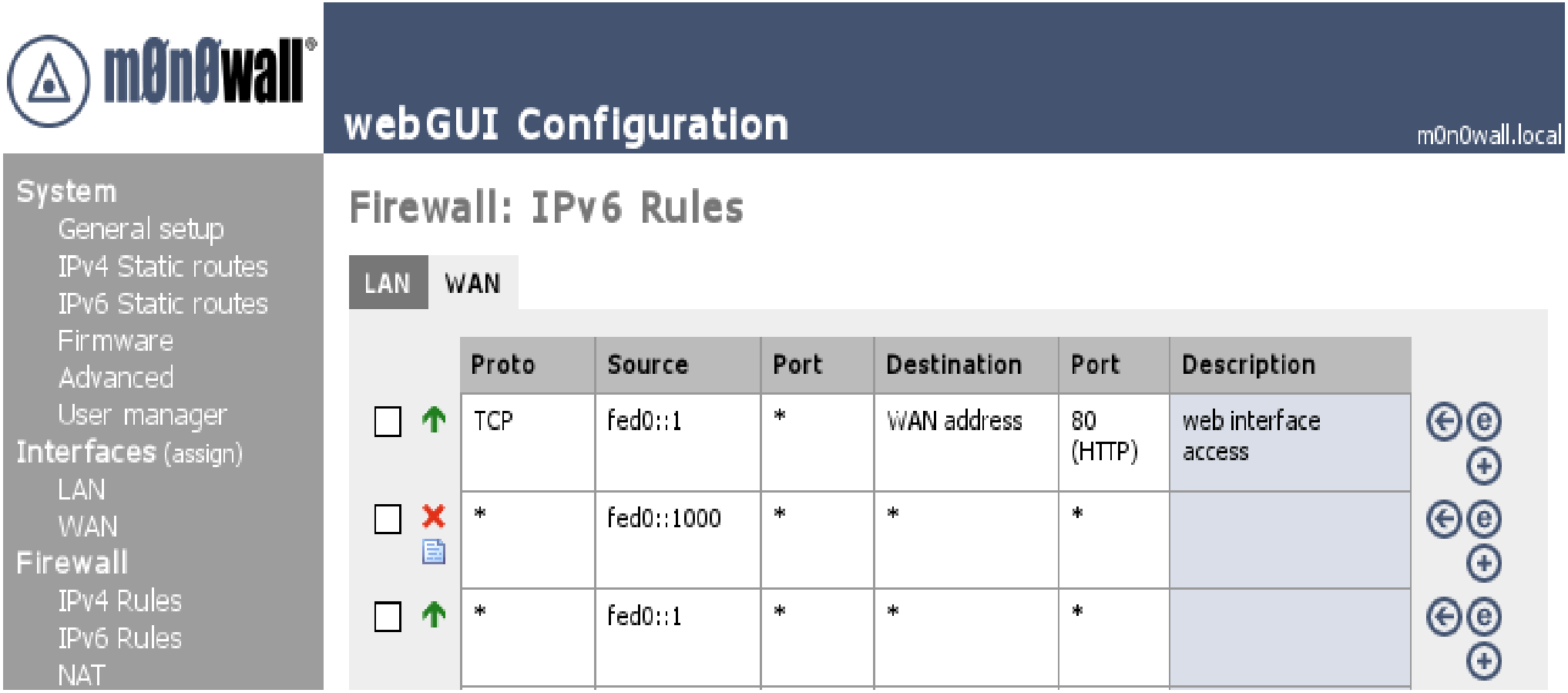
m0n0wall Rules



Firewall Evasion

- Demo 8

There was a small trick though



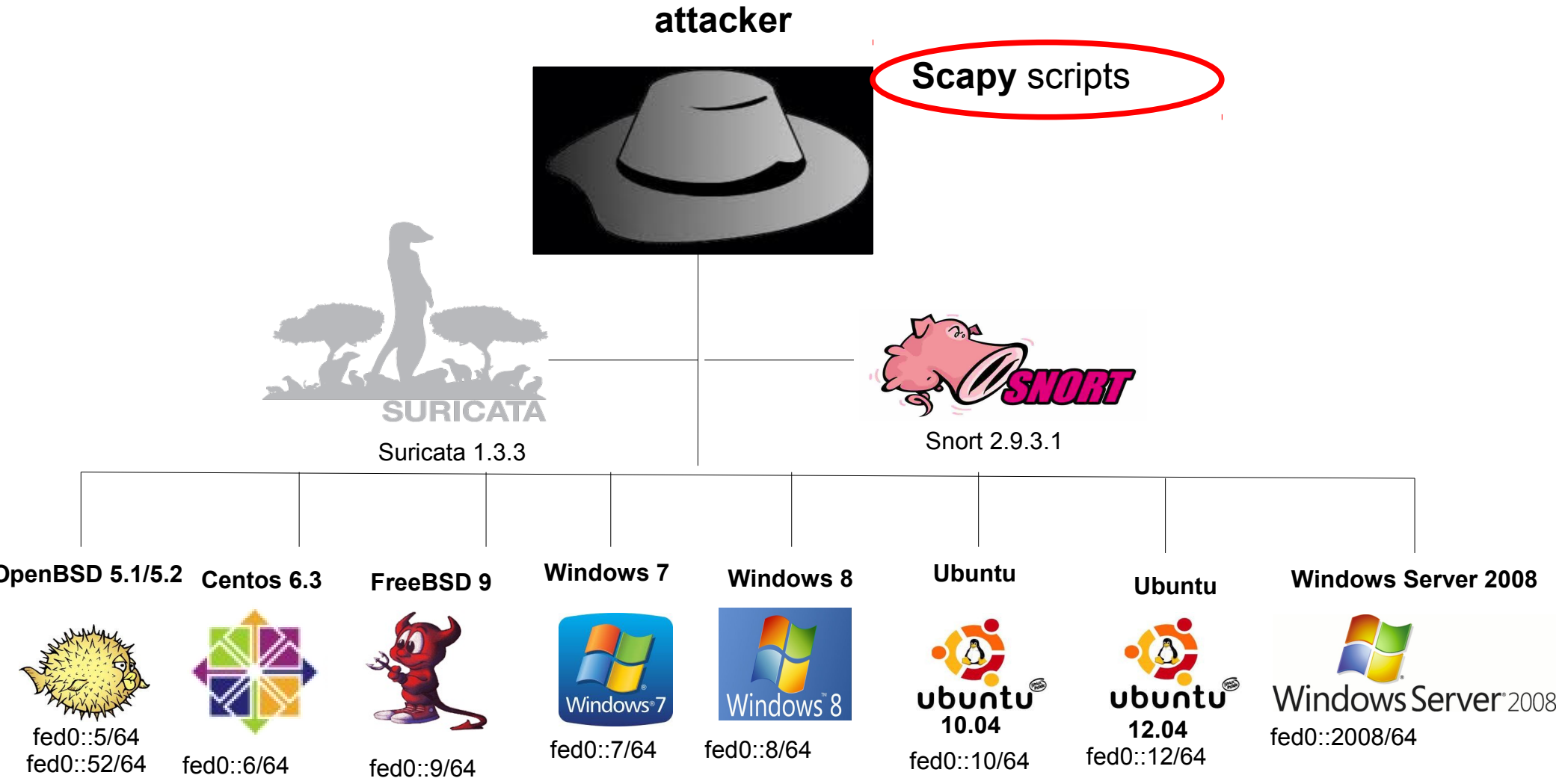
Evading IDS

- **IDS evasion**: When the end-system accepts a packet that the IDS (for some reason) rejects.
  - Hence, IDS misses the content of such a packet entirely, resulting in slipping through the IDS.
- **IDS insertion**: an IDS accepts a packet that the end-system rejects.
  - If properly manipulated, IDS signatures can also be defeated.

Evading IDS

- We shall “exploit” the IPv6 Extension Header abuse to evade IDS.
- Snort and Suricata were tested.
- An ICMPv6 Echo Request detection rule was enabled.
- Goal. Send ping6 and get a reply back from a target without being detected by the IDS.

The Lab Environment

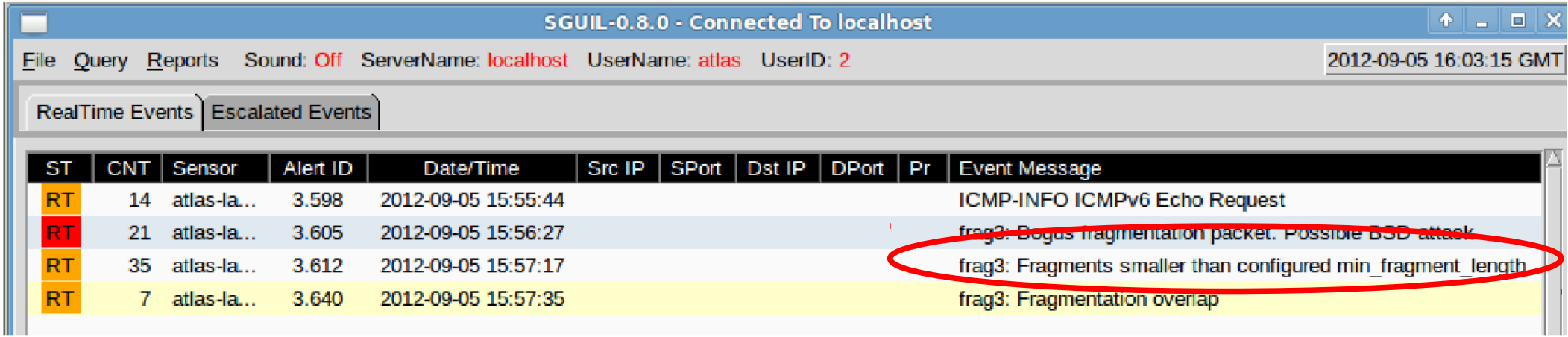


Let's try some attacks

- Demo 9

Tests	Alert(s) issued by Snort IDS
1. Mixing Multiple and Various Extension Headers per datagram in atomic fragments	~~frag:3: Bogus fragmentation packet. Possible BSD attack~~ ICMP-INFO ICMPv6 Echo Request
2.Nested fragments	~~frag:3: Bogus fragmentation packet. Possible BSD attack~~ frag3: Fragments smaller than configured min_fragment_length ICMP-INFO ICMPv6 Echo Request
3.Upper-layer Protocol Header at the Second/Subsequent Fragment	ICMP-INFO ICMPv6 Echo Request (for less than 8 fragments)
4. Upper-layer Protocol Header at the Second Fragment and Mixing Multiple Extension headers at the 1st	ICMP-INFO ICMPv6 Echo Request
5 Upper-layer Protocol Header at the 2nd Fragment with Extension Headers Overlapping	~~frag:3: Bogus fragmentation packet. Possible BSD attack~~ frag 3: Fragmentation overlap ICMP-INFO ICMPv6 Echo Request
6 Upper-layer Protocol Header at the Third Fragment with the 3rd fragment overlapping the 2nd	frag 3: Fragmentation overlap frag 3: Fragments smaller than configured min_fragment length
7 Upper-layer Protocol Header at the Third Fragment with the 3rd fragment overlapping the 1st	frag 3: Fragmentation overlap ~~frag 3: Bogus fragmentation packet. Possible BSD attack~~ frag 3: Fragments smaller than configured min_fragment length ICMP-INFO ICMPv6 Echo Request
8 Transfer of "large" amount of arbitrary data at the IP level	ICMP-INFO ICMPv6 Echo Request
9 Transfer of "large" amount of fragmented arbitrary data at the IP level	ICMP-INFO ICMPv6 Echo Request

Evading Snort



- One of the triggered alerts is the “fragment smaller than configured min_fragment_length”.
- This is due to the fact the each fragment has a very small amount of data in it (actually 1 octet), because it carries only the Destination Option Extension header.
- However, this can be avoided easily by adding arbitrary data as options in each one of these.

Evading Snort

- In case where the upper-layer protocol is sent at a fragment other than the first (case 3), we start to increase progressively the number of the fragments.

Evading Snort

```
for i in range(0,no_of_fragments):
      packet = IPv6(src=sip,dst=dip) \
            /IPv6ExtHdrFragment(offset=i*16,m=1) \
            /IPv6ExtHdrDestOpt(nh=60, options=PadN(optdata='\101'*120))
      send(packet)
packet = IPv6(src=sip,dst=dip) \
        /IPv6ExtHdrFragment(offset=no_of_fragments*16,m=1) \
        /IPv6ExtHdrDestOpt(nh=58, options=PadN(optdata='\101'*120))
send(packet)
packet = IPv6(src=sip,dst=dip) \
        /IPv6ExtHdrFragment(offset=(no_of_fragments+1)*16,m=0,nh=58) \
        /ICMPv6EchoRequest()
send(packet)
```

Evading Snort

Demo 10

- If we send the upper-layer header at 10th packet or later
- And fill the Destination Options Header with some arbitrary meaningless data at the options:
  - the ICMPv6 Echo Request message is not detected by Snort (an alert is not issued).
  - OpenBSD, Windows and Linux happily respond with an ICMPv6 Echo Reply message.

Evading Snort

- Using this same type of attack, we can launch any type of attack without being detected by Snort.
  - Port scanning, SQLi, etc.

Evading Snort

- As a proof-of-concept, we tried to avoid any detection when using smb activity.

  alert tcp any any -> any 135 (msg: "Test TCP activity at port 135"; sid:1000001;)

- We can also add some data into the SYN packet, which normally triggers a "*stream5: Data on SYN packet*" alert and still avoid detection.

- A quick demo (demo 11).

Evading Suricata

- Tested and configured similarly as Snort.
- Suricata-specific IPv6 rules were also enabled.
- Regarding the rest, the same ICMPv6 detection rule were enabled.

Tests	Alert(s) issued by Suricata IDS
1. Mixing Multiple and Various Extension Headers per datagram in atomic fragments	~~SURICATA IPv6 duplicated Destination Options extension header~~ ICMP-INFO ICMPv6 Echo Request
2. Nested fragments	~~NONE~~ SURICATA ICMPv6 invalid checksum ICMP-INFO ICMPv6 Echo Request
3.Upper-layer Protocol Header at the Second/Subsequent Fragment	~~NONE~~ SURICATA ICMPv6 invalid checksum ICMP-INFO ICMPv6 Echo Request (tested for 180 fragments)
4. Upper-layer Protocol Header at the Second Fragment and Mixing Multiple Extension headers at the 1st	~~NONE~~ SURICATA ICMPv6 invalid checksum ICMP-INFO ICMPv6 Echo Request
5 Upper-layer Protocol Header at the 2nd Fragment with Extension Headers Overlapping	SURICATA FRAG IPv6 Fragmentation overlap ICMP-INFO ICMPv6 Echo Request
6 Upper-layer Protocol Header at the Third Fragment with the 3rd fragment overlapping the 2nd	SURICATA FRAG IPv6 Fragmentation overlap SURICATA ICMPv6 invalid checksum
7 Upper-layer Protocol Header at the Third Fragment with the 3rd fragment overlapping the 1st	~~NONE~~ ICMP-INFO ICMPv6 Echo Request
8 Transfer of "large" amount of arbitrary data at the IP level	ICMP-INFO ICMPv6 Echo Request
9 Transfer of "large" amount of fragmented arbitrary data at the IP level	~~NONE~~ frag 3: Fragmentation overlap SURICATA ICMPv6 invalid checksum

Regarding Detection of IPv6 Tunneled in IPv6

	Windows XP	Snort	Suricata	Notes
IPv6 Tunnelled in IPv6	Y	- ET Policy 41 IPv6 encapsulation potential 6in4 IPv6 tunnel active	- ET Policy 41 IPv6 encapsulation potential 6in4 IPv6 tunnel active - POLICY-OTHER IPv6 packets encapsulated in IPv4	
Fragmented IPv6 Tunnelling	Y	-	- ET Policy 41 IPv6 encapsulation potential 6in4 IPv6 tunnel active - POLICY-OTHER IPv6 packets encapsulated in IPv4 - SURICATA IPv6 useless Fragment extension header	For 10 fragments, Snort is evaded! Or for 3 tunneled headers, 3 fragments, snort is evaded.
2nd way of fragmented Tunnel of IPv6 in IPv6	Y	-	- POLICY-OTHER IPv6 packets encapsulated in IPv4	for 3 tunneled headers, 3 fragments, snort is evaded.
3rd way of Fragmented IPv6 Tunnelling	Y	-	- ET Policy 41 IPv6 encapsulation potential 6in4 IPv6 tunnel active - POLICY-OTHER IPv6 packets encapsulated in IPv4	

Other Security Implications

- Unnecessarily use of IPv6 Extension Headers can be used to circumvent the **RA-Guard** protection.
  - When layer-2 devices check only the next-field of the base IPv6 Header to detect an ICMPv6 Router Advertisement message.
  - Fragmentation of the IPv6 Header Chain may make the situation more complicated and circumvent easier layer-2 devices.

Proposed Countermeasures

- RFCs should strictly define:
  - the exact usage and order of the IPv6 Extension headers
  - the respective OS response in case of non-compliant IPv6 datagrams.
- OS or security devices vendors should create fully RFC compliant products and test them thoroughly before claiming IPv6 readiness.

Proposed Countermeasures

- Security devices such as IDS/IPS and Data Loss Prevention (DLP) devices should be able to examine:
  - Not only “usual” IP attacks like IP fragmentation overlapping attacks, but also, new attacks which may exploit the new features and functionality of IPv6.
  - Not just the payload of the application layer protocols, but also the data transferred in the IPv6 Extension headers too.

Proposed Countermeasures

- "Quick and dirty" Solutions:
  - Prevent the acceptance of some of the IPv6 Extension headers using proper firewall rules.
  - Should be considered only as temporary ones, since they actually suppress some of the IPv6 added functionality and thus, should be applied only after ensuring that this functionality is actually not needed in the specific environment.
  - For example, can we suppress Fragment Extension Headers?

Conclusions

- IPv6 Extension headers add features and flexibility.
- But they also create new attack vectors.

Conclusions

- Various combinations of malformed (regarding the usage of the IPv6 Extension headers) IPv6 packets are accepted by most (if not all) the popular OS (including enterprise/servers or workstations).
- FreeBSD appears to have the most robust and RFC-compliant behaviour.
- Ubuntu/WinXP appears to have the worst.

Conclusions

- Very popular users' workstations or enterprise OS were found to be vulnerable to most of the examined malformed packets.
- Proper exploitation can lead to:
  - OS Fingerprinting
  - Covert channels
  - Firewall Evasion
  - IDS Evasion at the IP level
    - Using a single attack method allows attacks from port scanning to SQLi, without being detected by the corresponding IDS signatures.

Related draft-RFCs

- **Security and Interoperability Implications of Oversized IPv6 Header Chains**
  - *"If an IPv6 packet is fragmented, the first fragment of that IPv6 packet (i.e., the fragment having a Fragment Offset of 0) MUST contain the entire IPv6 header chain.*
  - *A host that receives an IPv6 first-fragment that does not contain the entire IPv6 header chain SHOULD drop that packet, and also MAY send an ICMPv6 error message to the (claimed) source address."*

Related draft-RFCs

- **Security and Interoperability Implications of Oversized IPv6 Header Chains**
  - But is this the proper way of handling IPv6 Header Chains?
  - Definitely more secure, but will this reduce the features that IPv6 may offer?
  - For instance, the size of an IPv6 Destination Option header can be up to 2048 bytes, and we can have two of them, plus a Hop-by-hop extension header (with the same size) plus any other IPv6 Extension headers.
- This is an issue open for discussion...

The Goal of This Presentation

- Not to show just a few tricks by abusing IPv6 for security impacts.
- IPv6 is a complex protocol. Crafting packets in a non-predicting ways may trigger really surprisingly results.
- Not all the IPv6 Extension Headers and their usage tested.
- Just some representative OS tested. Not mobile devices, not commercial networking or security devices. How about them?
- Several draft RFCs on the way. It seems that still a lot has to be done, though.
- Imagination is your limit.

Questions?

- Email: antonios.atlasis@gmail.com
  antonios.atlasis@cscss.org