Chiron

An All-In-One IPv6 Pen-Testing Framework

Antonios Atlasis, aatlasis@secfu.net

A Brief Bio

- An IT engineer for more than 20 years, developer and instructor in several Computer Science and Computer Security related fields.
- Penetration tester, incident handler and intrusion analyst and cyber-researcher for the last 7 years.
- MPhil (University of Cambridge), PhD (National Technical University of Athens).
- More than 25 scientific papers published in several international Journals and Conferences.
- Several GIAC certifications (GCIH, GWAPT, GREM, GPEN, GCIA and GXPN), and a Giac Gold Adviser (having supervised more than 20 Giac Gold papers).
- Latest security researching interests: IPv6, IDS/IPS and WAF evasions, SCADA systems. Some of the work has been presented at BlackHat Europe 2012, BlackHat Abu Dhabi 2012 and Troopers 13 security conferences.
- Currently working as an independent IT security analyst/consultant. Can reach me at **aatlasis@secfu.net**

Outline of the Presentation

- Introduction to the framework
- Generic parameters (available to most or even all modules)
- Network Scanning
  - Link-Local
  - Global (LAN/WAN)
- Sending Arbitrary IPv6 Neighbor Discovery Messages
- An IPv4-to-IPv6 Proxy
- Advanced IPv6 Scanning Techniques
  - Performing simple fragmentation
  - Flooding
  - Fuzzing (manually) IPv6 Extension Headers
  - IDS Evasion Techniques

Hints

- During the presentation, three tokens will be displayed: Write them down.
- During the workshops, you will be given several tasks.
- At the end, a challenge will also be given.

Why and How This Tool Was Build

- There are already great IPv6 Security tools.
- I always needed to do tests that were not covered by existing tools.
- I started building my own scripts using Scapy. Effective but not efficient because I had to write/change code for every single case that I wanted to change/test.
- So, I tried to build a tool that will you give all the flexibility you need to craft arbitrary IPv6 packets to run your own tests, not covered by other tools yet, but without having to write a single line of code.
- This is how *Chiron* was born. And I decided to share it with you.

This is the 1st Public Release of The Tool

- So, please, be patient, since we may encounter a few problems (aka, bugs).
- Keep notes of bugs, suggestions, features that you want me to add, etc.
- I am always open to comments and discussion.

Brief Introduction

- ***Chiron*** is written in Python; it uses *Scapy*, a very powerful Python library.
- It incorporates its own IPv6 sniffer(s).
- It is a mutli-threaded tool.
- It does not use the OS stack but Scapy libraries.
- A Framework not suitable for script-kiddies (you have to know IPv6 – RFCs are the manual...).
- Main advantage: You can easily craft arbitrary IPv6 header chain by using various types of IPv6 Extension Headers. This option can be used:
  - To evade IDS/IPS devices, firewalls, or other security devices.
  - To fuzz IPv6-capable devices regarding the handling of IPv6 Extension Headers.
- Main disadvantage: Many times you cannot stop it easily. You have to “kil”l it. This is because how python handles threads.

Main Modules (up to know)

- *IPv6 Scanner*
- *IPv6 Neighbor Discovery Messages Tool*
- *IPv4-to-IPv6 Proxy*
- *IPv6 Auto-Attacking Tool* (implementation in the near future)

All the above modules are supported by a common library that allows the creation of completely arbitrary IPv6 header chains, fragmented or not.

Main Characteristics

- Flexibility
- Modularity
- Fast performance (due to multi-threading), especially when a delay is introduced due to networking operations.
- Expandability
- But (deliberately) a tool not suitable for Script-Kiddies (you have to know IPv6 and how to use it).

Preparation

- To run the IPv6 Scanner, you need a slightly patched version of ***Scapy***, and of course, Python (version 2.7.x).
- Patched version of Scapy provides:
  - An IPv6 Fake Extension Header
  - A Bug Fix regarding the handling of *ICMPv6 Fragment Reassembly Time Exceeded*
- Download Chiron and Scapy from the Web Server
- If you use Windows, download VirtualBox and import Linux virtual appliance.
- Optionally, install the following python libraries:
  - python-crypto
  - PyX
  - gnuplot-py
- Then, build and install scapy:
  - *$ python setup.py build*
  - *# python setup.py install*

Task 0: Prepare Your Machine

- Prepare your attacking environment.
  - If you use Linux, simply download Chiron and Scapy from the lab web-server, or copy them
    - I assume that you already have Python installed.
    - A sniffer tcpdump/Wireshark would also be useful.
  - If you use Windows or MAC OS X:
    - Download and install VirtualBox
    - Import a provided Linux machine (which includes everything that you need).
    - Put the interface on Bridged mode. Ask me to help you if you do not know how!
  - USB flash drives are available with everything that you need.
- Web server address:

http://[2001:db8:c001:babe:224:54ff:feba:a197]/

The Tools

- All the tools are located into the ./bin directory:
  - ***chiron_scanner.py*** A network scanner
  - ***chiron_combinations.py*** For generating IPv6 suffixes by combining several words – useful for “smart” scanning
  - ***chiron_nd.py*** For generating arbitrary Neighbor Discovery Messages
  - ***chiron_proxy.py*** A multi-threaded IPv4-to-IPv6 proxy.
- The libraries are located into the ./lib directory (but you don't need to access them directly).

How to Use it

- You must run the binaries **as root**.
- **You must define at least the interface to use** (e.g. *./chiron_scanner.py eth0*, etc., depending on your OS).
- To use some of the advanced features, you must also patch Scapy (patching files are provided).
- IMPORTANT NOTE: While running (at least the advanced techniques of) the IPv6 Scanner, please make sure not to run any other IPv6 activities (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.

Generic Parameters

Common to all the available modules of the framework

First, Define the Network Interface to Use

- Example:

  ./chiron_scanner.py eth0   ...etc.

  (depending on your OS)

Defining The Targets

Defining Your Targets

- Available the following options:
  - A comma separated list of IPv6 addresses or FQDN (CLI)
  - A range of IPv6 addresses
  - IPv6 subnets (but be careful, if you want to finish in this ...life)
  - A list of IPv6 addresses or FQDN in a text file (one per line).
  - Automatic combinations of suffixes of your choice with a chosen IPv6 prefix.

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).

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 info in the Tutorial).
- 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.
- You can use such combinations with /64 prefixes.
- You can create a list of such possible combinations using the following steps.

“Smart” Scan

1. Create a list of potential suffixes in a text file, one per line, e.g.:

   face

   b00c

   beef

   ...etc.

- Sample files are given in the files directory (named *combinations.txt* and *combinations-small.txt*).

“Smart” Scan

2. Use the ***./chiron_combinations.py*** binary to create the possible combinations. Example:

```
./chiron_combinations.py ../files/combinations-small.txt output.txt
```

Sample of the output file:

```
:face:b00c:f00d:abba
:face:b00c:f00d:b00b
:face:b00c:f00d:b0b0
:face:b00c:f00d:babe
:face:b00c:f00d:bead
:face:b00c:f00d:beef
```

...etc.

You have to create such a file just once (unless you want to add more suffixes).

“Smart” Scan

3. Once you have created, you just need to use it using the *-sM* switch, but, moreover:

-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 vboxnet0 -sM -pr fdf3:f0c0:2567:7fe4
-iC ../files/my_combinations-small.txt -sn
```

Defining (or Spoofing) The Source Addresses

Defining (spoofing) source addresses

- The source address of your packets is chosen is following:
  - If an IPv6 source and a MAC source addresses are not defined, your machine's IPv6 address and the corresponding MAC address are used of the interface you have chosen.
  - If you randomise or define (spoof) a source MAC address, your IPv6 address and the spoofed MAC address are used.
  - If you define (spoof) just a source IPv6 address, the corresponding MAC address is used as a source (it is found using Neighbor Solicitation - NS). If NS does not return a MAC address, a random MAC address is used.
  - If you spoof or randomise both the IPv6 address and the MAC address, these specific spoofed MAC addresses are used.

Defining (spoofing) source addresses

- Switches to use:
  - -s <IPv6 source address> The IPv6 address you want to specify as a source address.
  - -m <MAC 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.
  - -rm   Randomise the source MAC address. You do not have to define anything else.

(Some) Other Generic Parameters

- -hoplimit <Hop Limit> Values: 0 to 255. Default values: 64 for the scanner, 255 for the neighbor discovery (nd) tool.
- -threads <NO_OF_THREADS > The number of threads to use (for multi-threaded operation). Default value: 10
- Store the results to a file:

  -of <OUTPUT_FILE>   The filename where the results will be stored (otherwise displayed at the stdout).

Other Various (Generic) Parameters

- During the various scanning/attack methods, the following switches can also be used that either provides more info, or specialise some scamming details:

    -nsol Display neighbor solicitation results (IPv6 vs MAC addresses) for your info. Default: False.

    -timeout <SEND_TIMEOUT> The timeout (in seconds) if resending of packets is required (for instance when a response has not been received). Default value: 2

    -no_of_retries <NUMBER_OF_RETRIES> The number of retries when a response is not received. Default value: 2

    -stimeout <SNIFFER_TIMEOUT> The timeout (in seconds) when the integrated sniffer (IF used) will exit automatically. Default value: 60 seconds

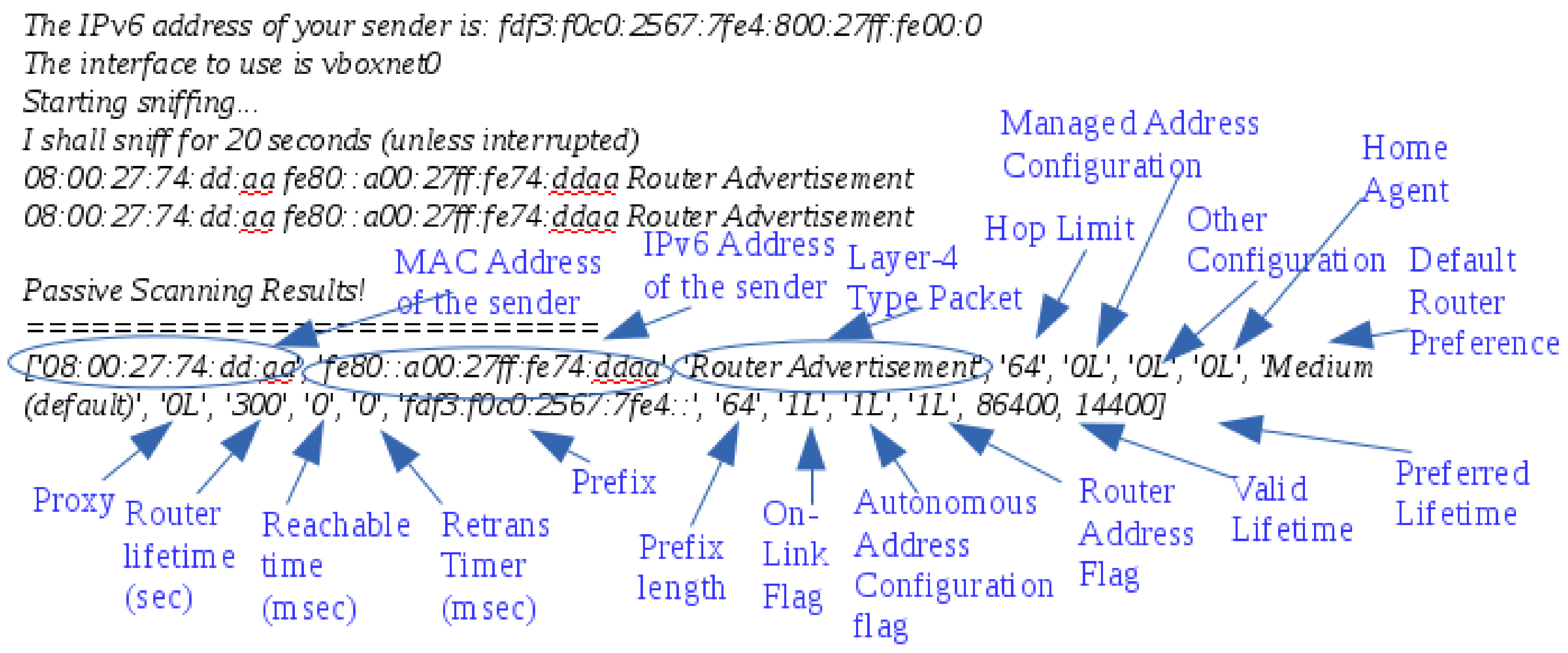
1. The IPv6 Scanner Module

Link-Local Scanning

Simple IPv6 Scanning

- -rec   Sniffs the wire passively (default: 10 seconds). Change the sniffing time using the -stimeout switch.

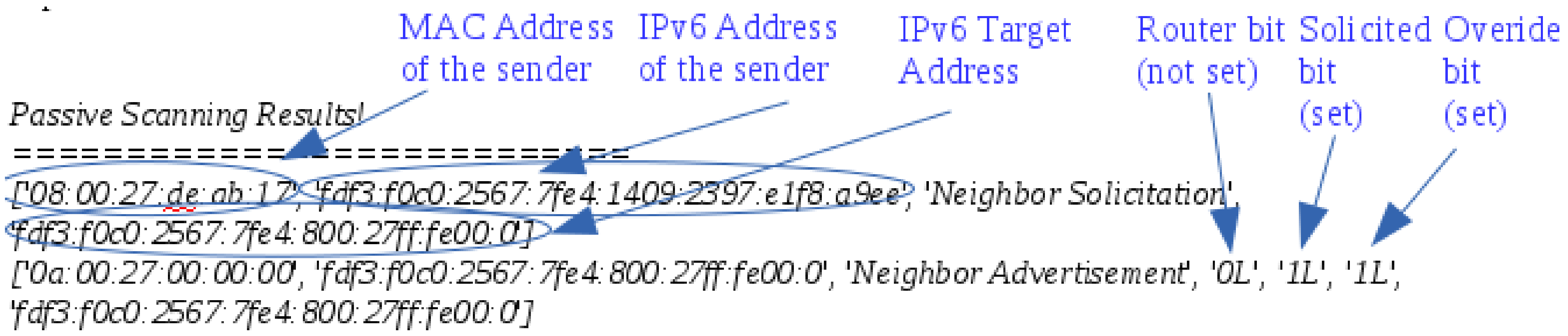
Passive Scanning – Example Output



```
The IPv6 address of your sender is: fdf3:f0c0:2567:7fe4:800:27ff:fe00:0
The interface to use is vboxnet0
Starting sniffing...
I shall sniff for 20 seconds (unless interrupted)
08:00:27:74:dd:aa fe80::a00:27ff:fe74:ddaa Router Advertisement
08:00:27:74:dd:aa fe80::a00:27ff:fe74:ddaa Router Advertisement

Passive Scanning Results!
===========================
['08:00:27:74:dd:aa', 'fe80::a00:27ff:fe74:ddaa', 'Router Advertisement', '64', '0L', '0L', '0L', 'Medium
(default)', '0L', '300', '0', '0', 'fdf3:f0c0:2567:7fe4::', '64', '1L', '1L', '1L', 86400, 14400]
```

Passive Scanning – Yet Another Example Output



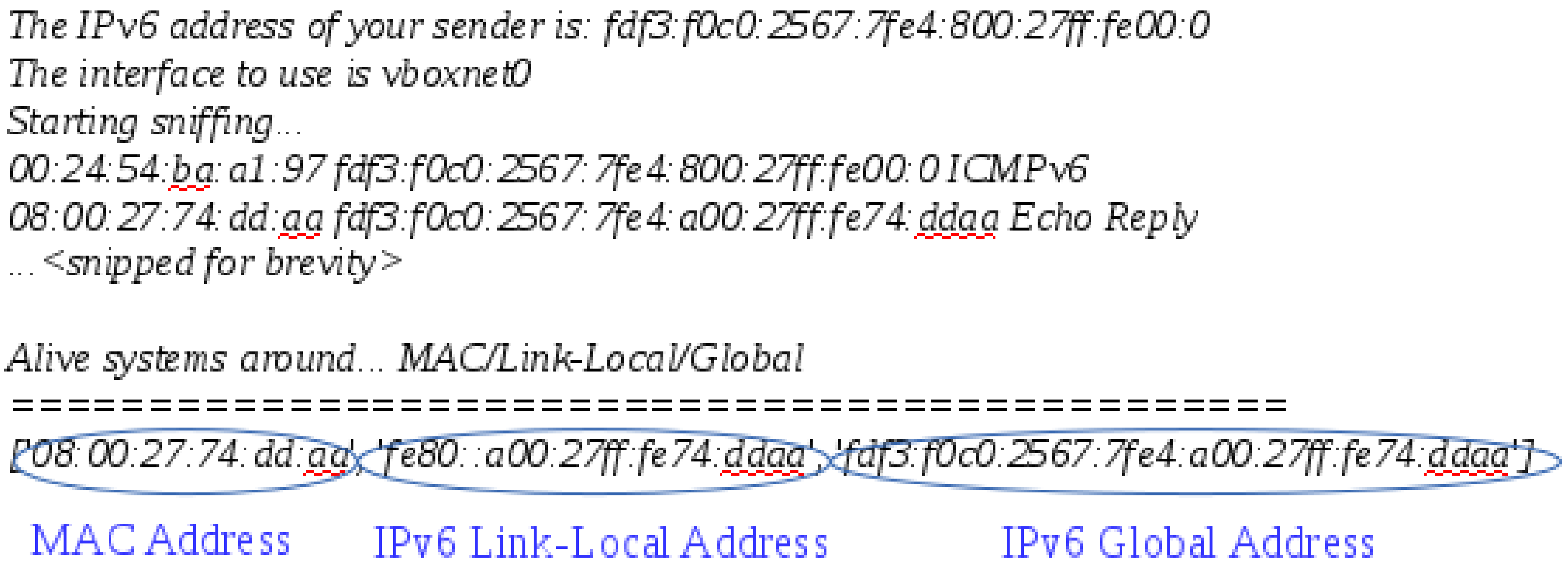
```
Passive Scanning Results!
==============================
['08:00:27:de:ab:17', 'fdf3:f0c0:2567:7fe4:1409:2397:e1f8:a9ee', 'Neighbor Solicitation',
'fdf3:f0c0:2567:7fe4:800:27ff:fe00:0']
['0a:00:27:00:00:00', 'fdf3:f0c0:2567:7fe4:800:27ff:fe00:0', 'Neighbor Advertisement', '0L', '1L', '1L',
'fdf3:f0c0:2567:7fe4:800:27ff:fe00:0']
```

Token 1: 8096-3485-6232-3080-8540

Simple IPv6 Scanning

- -rec Sniffs the wire passively (default: 10 second). Change the sniffing time using the -stimeout switch.
- -mpn Multicast ICMPv6 Scan
  - Messages are sent to IPv6 address "ff02::1" and MAC address "33:33:00:00:00:01".
  - The following messages are sent:
    - A legitimate ICMPv6 Echo Request
    - An Unsolicited Neighbor Advertisement
    - An ICMPv6 Echo Request preceded by an IPv6 Destination Options Header with an unknown Option (to trigger an "*ICMPv6 Parameter Problem - unrecognized IPv6 Option encountered*").
    - An ICMPv6 Echo Request preceded by an non-existing (Fake) IPv6 Extension Header (to trigger an "*ICMPv6 Parameter Problem - unrecognized Next Header type encountered*")

Multicast Ping Scan – Example Output



```
The IPv6 address of your sender is: fdf3:f0c0:2567:7fe4:800:27ff:fe00:0
The interface to use is vboxnet0
Starting sniffing...
00:24:54:ba:a1:97 fdf3:f0c0:2567:7fe4:800:27ff:fe00:0 ICMPv6
08:00:27:74:dd:aa fdf3:f0c0:2567:7fe4:a00:27ff:fe74:ddaa Echo Reply
...<snipped for brevity>

Alive systems around... MAC/Link-Local/Global
=========================================
['08:00:27:74:dd:aa', 'fe80::a00:27ff:fe74:ddaa', 'fdf3:f0c0:2567:7fe4:a00:27ff:fe74:ddaa']
```

Simple IPv6 Scanning

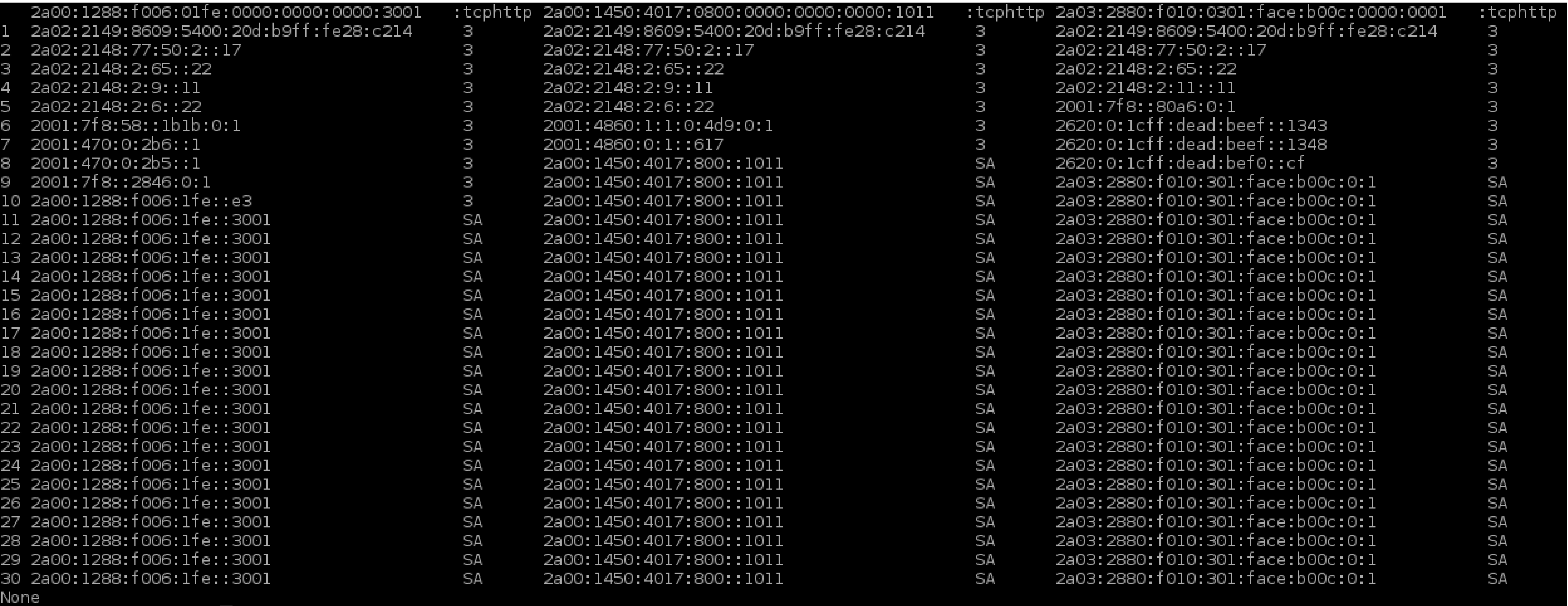
- -rec Sniffs the wire passively (default: 10 second). Change the sniffing time using the -stimeout switch.
- -mpn Multicast ICMPv6 Scan
  - Messages are sent to IPv6 address "ff02::1" and MAC address "33:33:00:00:00:01".
  - The following messages are sent:
    - A legitimate ICMPv6 Echo Request
    - An Unsolicited Neighbor Advertisement
    - An ICMPv6 Echo Request preceded by an IPv6 Destination Options Header with an unknown Option (to trigger an ICMPv6 Parameter Problem - unrecognized IPv6 Option encountered).
    - An ICMPv6 Echo Request preceded by an non-existing (Fake) IPv6 Extension Header (to trigger a to trigger an ICMPv6 Parameter Problem - unrecognized Next Header type encountered)
- -sn Ping Sweep

IPv6 Tracerouting

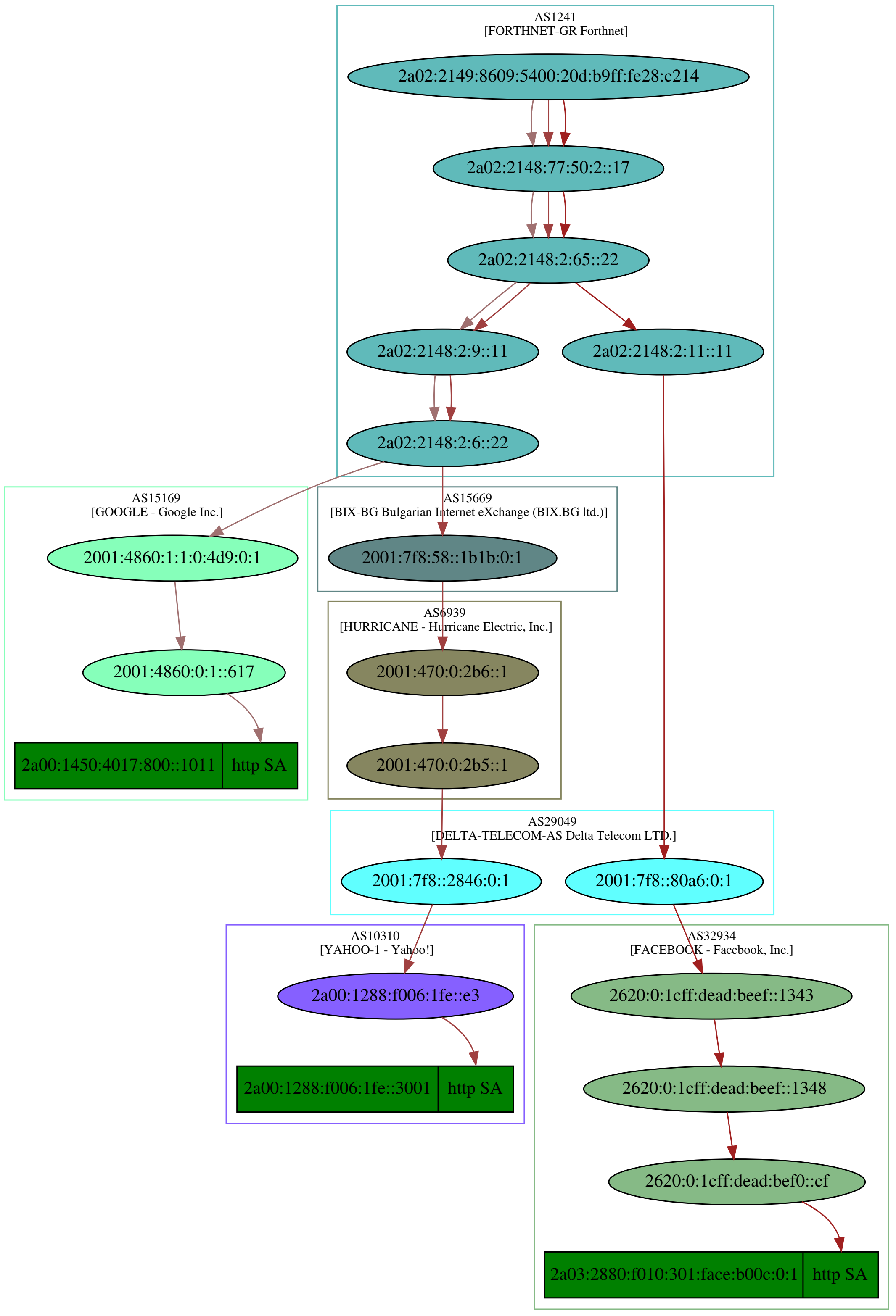
-tr-gr	tracerouting graph
-tr	generic tracerouting

Simple IPv6 TCP Tracerouting

- Use -tr-gr
- You have to define your destinations only in a comma separated list using the -d switch.
- It sends all the packets at the same time, in parallel.

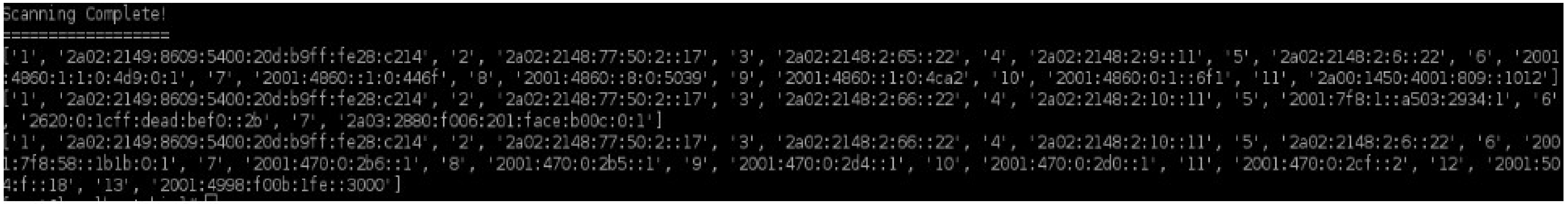


```
   2a00:1288:f006:01fe:0000:0000:0000:3001   :tcphttp 2a00:1450:4017:0800:0000:0000:0000:1011   :tcphttp 2a03:2880:f010:0301:face:b00c:0000:0001   :tcphttp
1  2a02:2149:8609:5400:20d:b9ff:fe28:c214     3       2a02:2149:8609:5400:20d:b9ff:fe28:c214     3       2a02:2149:8609:5400:20d:b9ff:fe28:c214     3
2  2a02:2148:77:50:2::17                      3       2a02:2148:77:50:2::17                      3       2a02:2148:77:50:2::17                      3
3  2a02:2148:2:65::22                         3       2a02:2148:2:65::22                         3       2a02:2148:2:65::22                         3
4  2a02:2148:2:9::11                          3       2a02:2148:2:9::11                          3       2a02:2148:2:11::11                         3
5  2a02:2148:2:6::22                          3       2a02:2148:2:6::22                          3       2001:7f8::80a6:0:1                         3
6  2001:7f8:58::1b1b:0:1                      3       2001:4860:1:1:0:4d9:0:1                    3       2620:0:1cff:dead:beef::1343                3
7  2001:470:0:2b6::1                          3       2001:4860:0:1::617                         3       2620:0:1cff:dead:beef::1348                3
8  2001:470:0:2b5::1                          3       2a00:1450:4017:800::1011                   SA      2620:0:1cff:dead:bef0::cf                  3
9  2001:7f8::2846:0:1                         3       2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
10 2a00:1288:f006:1fe::e3                     3       2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
11 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
12 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
13 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
14 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
15 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
16 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
17 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
18 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
19 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
20 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
21 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
22 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
23 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
24 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
25 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
26 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
27 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
28 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
29 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
30 2a00:1288:f006:1fe::3001                   SA      2a00:1450:4017:800::1011                   SA      2a03:2880:f010:301:face:b00c:0:1           SA
None
```


AS1241
[FORTHNET-GR Forthnet]
2a02:2149:8609:5400:20d:b9ff:fe28:c214
2a02:2148:77:50:2::17
2a02:2148:2:65::22
2a02:2148:2:9::11
2a02:2148:2:11::11
2a02:2148:2:6::22
AS15169
[GOOGLE - Google Inc.]
2001:4860:1:1:0:4d9:0:1
2001:4860:0:1::617
2a00:1450:4017:800::1011
http SA
AS15669
[BIX-BG Bulgarian Internet eXchange (BIX.BG ltd.)]
2001:7f8:58::1b1b:0:1
AS6939
[HURRICANE - Hurricane Electric, Inc.]
2001:470:0:2b6::1
2001:470:0:2b5::1
AS29049
[DELTA-TELECOM-AS Delta Telecom LTD.]
2001:7f8::2846:0:1
2001:7f8::80a6:0:1
AS10310
[YAHOO-1 - Yahoo!]
2a00:1288:f006:1fe::e3
2a00:1288:f006:1fe::3001
http SA
AS32934
[FACEBOOK - Facebook, Inc.]
2620:0:1cff:dead:beef::1343
2620:0:1cff:dead:beef::1348
2620:0:1cff:dead:bef0::cf
2a03:2880:f010:301:face:b00c:0:1
http SA

Generic IPv6 Tracerouting

- Use the *-tr* switch.
- You can define your destinations in any of the ways described before.
- It can be combined with all the advanced / fuzzing techniques described later.



```
Scanning Complete!
==================
['1', '2a02:2149:8609:5400:20d:b9ff:fe28:c214', '2', '2a02:2148:77:50:2::17', '3', '2a02:2148:2:65::22', '4', '2a02:2148:2:9::11', '5', '2a02:2148:2:6::22', '6', '2001
:4860:1:1:0:4d9:0:1', '7', '2001:4860::1:0:446f', '8', '2001:4860::8:0:5039', '9', '2001:4860::1:0:4ca2', '10', '2001:4860:0:1::6f1', '11', '2a00:1450:4001:809::1012']
['1', '2a02:2149:8609:5400:20d:b9ff:fe28:c214', '2', '2a02:2148:77:50:2::17', '3', '2a02:2148:2:66::22', '4', '2a02:2148:2:10::11', '5', '2001:7f8:1::a503:2934:1', '6'
, '2620:0:1cff:dead:bef0::2b', '7', '2a03:2880:f006:201:face:b00c:0:1']
['1', '2a02:2149:8609:5400:20d:b9ff:fe28:c214', '2', '2a02:2148:77:50:2::17', '3', '2a02:2148:2:66::22', '4', '2a02:2148:2:10::11', '5', '2a02:2148:2:6::22', '6', '200
1:7f8:58::1b1b:0:1', '7', '2001:470:0:2b6::1', '8', '2001:470:0:2b5::1', '9', '2001:470:0:2d4::1', '10', '2001:470:0:2d0::1', '11', '2001:470:0:2cf::2', '12', '2001:50
4:f::18', '13', '2001:4998:f00b:1fe::3000']
```

- The results are presented in a line per-target.
- The number before the IPv6 address gives how many hops away is this address from the source node.

Generic IPv6 Tracerouting

- Optional Parameters:
  - -max_ttl <ttl> Define the maximum TTL to be used during the multi-parallel traverouting packets.
  - -min_ttl <ttl> Define the minimum TTL to be used during the multi-parallel traverouting packets.
  - -l4 <proto> The layer-4 protocol to be used for the tracerouting messages. Possible values: tcp, udp, icmpv6 (default)
  - -l4_data <proto_data> The data to be used as a layer 4 payload.

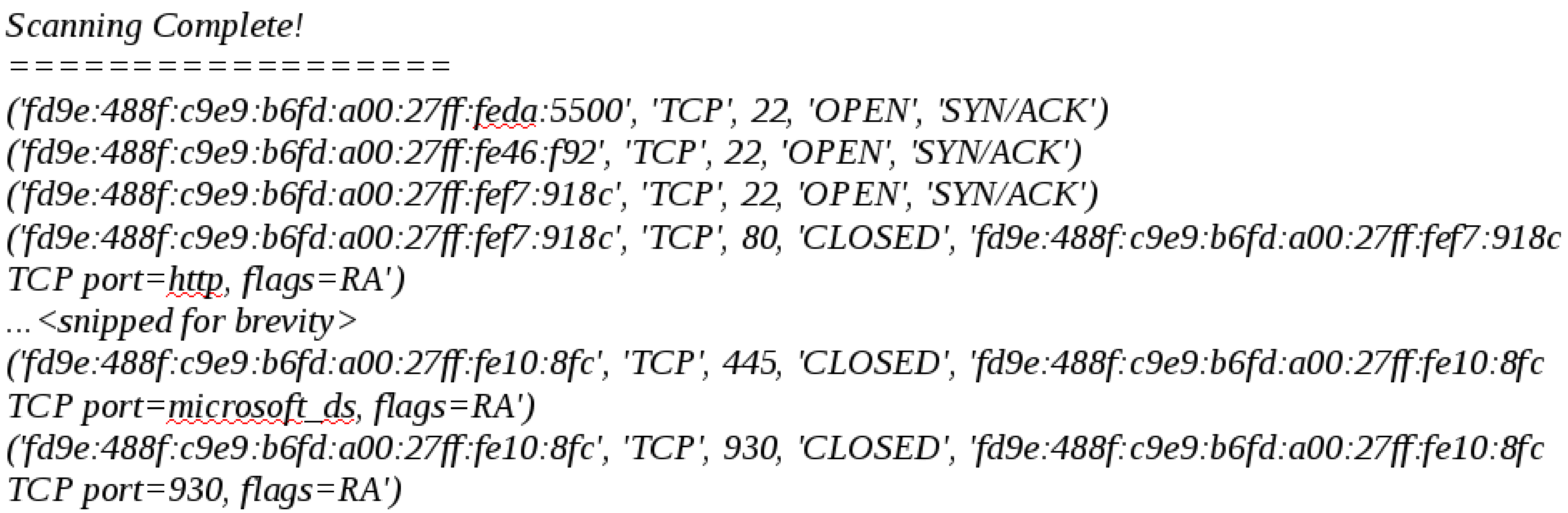
TCP / UDP Scanning

- -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
- Define your destination ports, using either a comma-separated list, or a range of ports or a combination of them using the -p switch.

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 range (if not defined): ports 1 to 1024
- Source ports are randomised per destination.

TCP Port Scanning – Example Output



```
Scanning Complete!
===================
('fd9e:488f:c9e9:b6fd:a00:27ff:feda:5500', 'TCP', 22, 'OPEN', 'SYN/ACK')
('fd9e:488f:c9e9:b6fd:a00:27ff:fe46:f92', 'TCP', 22, 'OPEN', 'SYN/ACK')
('fd9e:488f:c9e9:b6fd:a00:27ff:fef7:918c', 'TCP', 22, 'OPEN', 'SYN/ACK')
('fd9e:488f:c9e9:b6fd:a00:27ff:fef7:918c', 'TCP', 80, 'CLOSED', 'fd9e:488f:c9e9:b6fd:a00:27ff:fef7:918c
TCP port=http, flags=RA')
...<snipped for brevity>
('fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc', 'TCP', 445, 'CLOSED', 'fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc
TCP port=microsoft_ds, flags=RA')
('fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc', 'TCP', 930, 'CLOSED', 'fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc
TCP port=930, flags=RA')
```

- The results of a TCP port scanning can be, OPEN, CLOSED and FILTERED (no response).
- You get additional information when you get a response (OPEN/CLOSED cases)

UDP Port Scanning – Example Output

```
Scanning Complete!
==================
('fd9e:488f:c9e9:b6fd:a00:27ff:feda:5500', 'UDP', 80, 'CLOSED', 'fd9e:488f:c9e9:b6fd:a00:27ff:feda:5500 ICMPv6 Destination unreachable Communication with destination administratively prohibited')
('fd9e:488f:c9e9:b6fd:a00:27ff:feda:5500', 'UDP', 443, 'CLOSED', 'fd9e:488f:c9e9:b6fd:a00:27ff:feda:5500 ICMPv6 Destination unreachable Communication with destination administratively prohibited')
('fd9e:488f:c9e9:b6fd:a00:27ff:feda:5500', 'UDP', 445, 'CLOSED', 'fd9e:488f:c9e9:b6fd:a00:27ff:feda:5500 ICMPv6 Destination unreachable Communication with destination administratively prohibited')
```

IPv6-Specific Scanning Attacks

IPv6 Path MTU Discovery

Antonios Atlasis, aatlasis@secfu.net

IPv6 Path MTU Discovery

- A technique to dynamically discover the Path MTU (PMTU) of a path (RFC 1981).
  - A source node initially assumes that the PMTU of a path is the (known) MTU of the first hop in the path.
  - If any of the packets sent on that path are too large to be forwarded by some node along the path, that node will discard them and return ICMPv6 Packet Too Big messages. Upon receipt of such a message, the source node reduces its assumed PMTU for the path based on the MTU of the constricting hop as reported in the Packet Too Big message.
  - The Path MTU Discovery process ends when the node's estimate of the PMTU is less than or equal to the actual PMTU.

Source: RFC 1981

IPv6-Specific Scanning Attacks

- **Path MTU Discovery**

  -pmtu It performs Path MTU Discovery.

  -mtu The initial MTU to use for path MTU discovery (default=1500).

```
... <snipped for brevity>
Path MTU Discovery
------------------
sender= 2a02:2149:8602:7700:20d:b9ff:fe28:c214 PATH MTU = 1492
sender= 2a00:1450:4001:809::1013 PATH MTU = 1492
Scanning Complete!
===================
('2a02:2149:8602:7700:20d:b9ff:fe28:c214', 'ICMPv6 Packet Too Big', 'MTU=1492')
('2a00:1450:4001:809::1013', 'ICMPv6', 128, 'REACHABLE', '2a00:1450:4001:809::1013
2a02:2149:8602:7700:224:54ff:feba:a197 ICMPv6 Echo Reply (id=0xfdff)')
```

Task 1

- Find out “passively” (without generating any traffic) what there is around in the network regarding IPv6.
- Find out “alive” systems.
- Ping them (all in one command).
- Find out services that they offer.
  - Send the traffic only to alive systems
  - Experiment with the number of ports vs number of threads
- If we weren't in a LAN, we could also find al the hops to our targets, and the Path MTU to them.

IPv6 Scanner Examples

- **passive scanning:** *./chiron_scanner.py vboxnet0 -rec -stimeout 50*
- **multi-ping scan**: *./chiron_scanner.py vboxnet0 -mpn*
- **ping request**:

  ./chiron_scanner.py vboxnet0 -sn -d fdf3:f0c0:2567:7fe4:a00:27ff:fe74:ddaa
- **TCP SYN SCAN**:

  ./chiron_scanner.py vboxnet0 -d fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc -sS -p 22,443-445,80

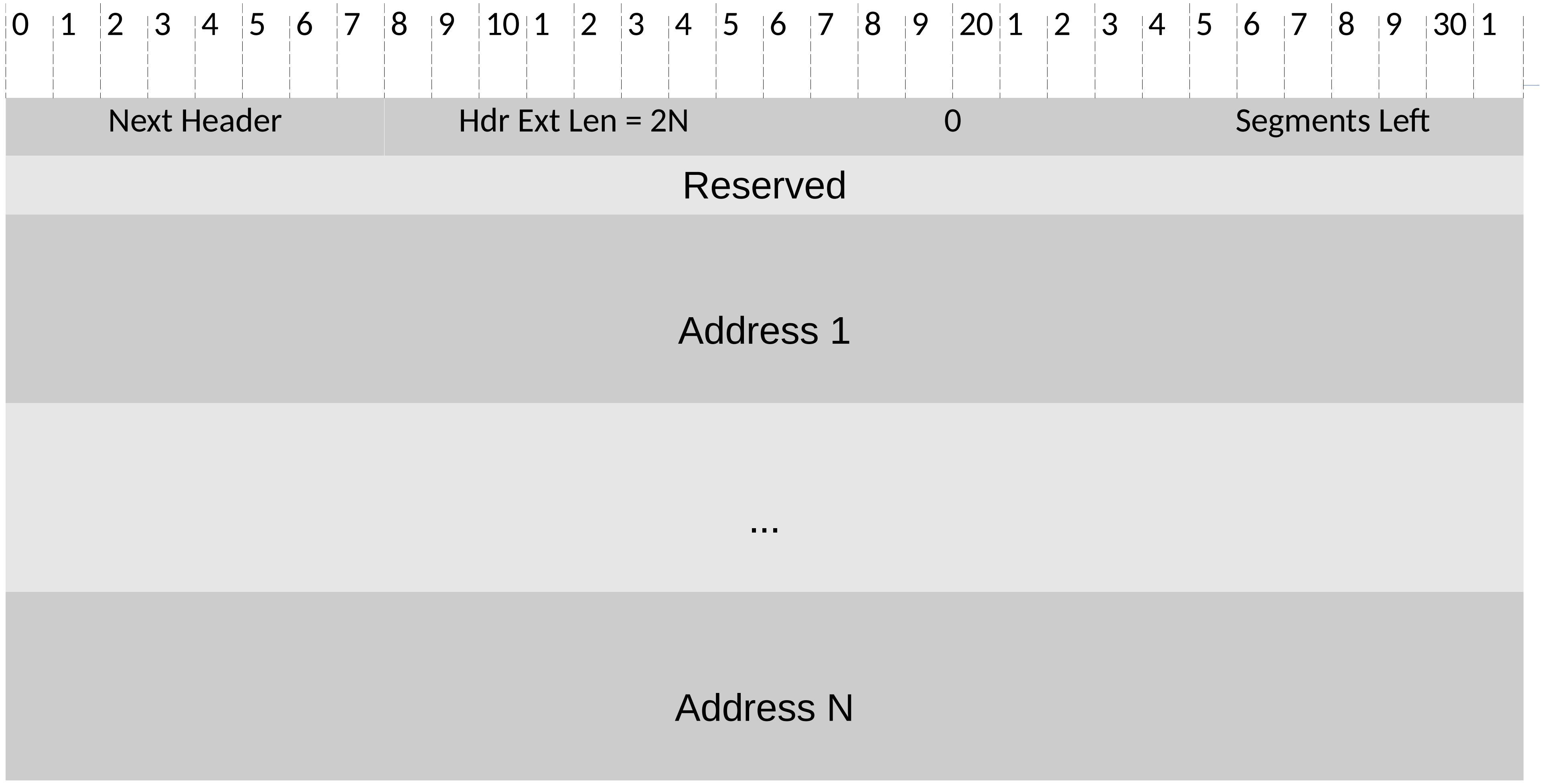
The IPv6 Routing Extension Header and the Type-0 Routing Extension Header

Antonios Atlasis, aatlasis@secfu.net

The IPv6 Routing Header

- Used by an IPv6 source to list one or more intermediate nodes to be "visited" on the way to a packet's destination.
- All IPv6 nodes must be able to process routing headers (nodes = routers + hosts).

The Type 0 Routing



- Equivalent to IPv4 lose source routing.
- Address N is the IPv6 address of the final destination, address 1, 2, 3, ..., N-1 are the IPv6 addresses of the intermediate routers.
- Routers **and hosts** process them.

Type 0 Routing Security Implications

- Firewall Evasion (e.g. if an intermediate target is allowed by a firewall, but the last one, “hided” in the Routing Header, is not).
- DoS Amplification attacks (by bouncing packets between two routers several times).
- Fortunately, with RFC 5095 in Dec 2007 Type 0 Routing Headers in IPv6 has been deprecated.

IPv6-Specific Scanning Attacks

- **Type 0 Routing Header Support Detection**

  -rh0

  -l4 <proto> The layer-4 protocol to be used. Possible values: *tcp, udp, icmpv6* (default).

  -l4_data <proto_data> The data to be used as a layer 4 payload.

2. Crafting Arbitrary Neighbor Discovery Messages

Crafting Arbitrary Neighbor Discovery Messages

- Router Advertisement Messages
- Router Solicitation Messages
- Neighbor Advertisement Messages
- Neighbor Solicitation Messages
- Router Redirect
- Packet Too Big

Router Advertisement Messages

- **RFC 4861**: They 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)

Potential Router Advertisement Attacks

- Send fake RA messages, using your machine's address, to potentially put you in the middle (you should also DoS the legitimate router).
- Spoof the IPv6 source address to DoS legitimate router by:
  - Setting Router lifetime = 0
  - Setting Router priority to Low (in combination with fake RA messages).
- Unset M/O flags: Implicitly DoS DHCPv6.

Token 2: 6600-7098-7032-1840-3109

Router Advertisement Messages

-ra	Send Router Advertisement (messages)
-chlim <Current Hop Limit>	Advertised Current Hop Limit - can be between 0 and 255. Default value for ND messages: 255
-M	Managed Address Configuration Flag. Default: False
-O	Other Configuration Flag. Default: False
-res <reserved>	Reserved field. Default Value: 0. Can be between 0 and 63
-pr <PREFIX>	The IPv6 prefix to use. Example: fe80:224:54ff:feba:: Default="fe80::”
-rl <ROUTER_LIFETIME>	The Router Lifetime - in seconds - for the Router Advertisement message - can be between 0 and 65535
-r_time <REACHABLE_TIME>	Reachable_time (in milliseconds) for Router Advertisement messages
-r_timer <RETRANS_TIMER>	Retrans timer (in milliseconds) for Router Advertisement messages
-rp <ROUTER_PRIORITY>	The Router Priority (default: high). Possible values 0: Medium 1: High 2: Reserved 3: Low
-pr-length <PREFIX_LENGTH>	The IPv6 prefix length to use
-mtu <DMTU>	The MTU value to use.

Router Solicitation Messages

-rsol Send Router Solicitation (messages)

-res <reserved> Reserved field. Default Value: 0

Neighbor Advertisement Messages

- **RFC 4861**:
  - They are sent out in response to NS or,
  - they are sent unsolicited in order to (unreliably) propagate information quickly.
- Some critical parameters:
  - ***Router*** flag: It indicates that the sender is a router
  - ***S*** flag: It indicates that the advertisement was sent in response to a Neighbor Solicitation message.
  - ***O*** flag: It indicates that the advertisement should override an existing entry and update the cached link-layer address.

Neighbor Advertisement Attacks

- Spoofed NA messages can be used for Neighbor cache poisoning in order to:
  - To launch DoS attacks
  - To launch MITM attacks.
  - To notify other recipients for a fake router, etc.

Neighbor Advertisement Messages

-neighadv Send neighbor advertisement messages. Default: False

-r Set the Router Flag for ICMPv6 Neighbor Advertisement messages. Default: False

-sol Set the Solicited Flag for ICMPv6 Neighbor Advertisement messages. Default: False

-o Set the Override Flag for ICMPv6 Neighbor Advertisement messages. Default: False

-ta <TARGET_ADDRESS> The IPv6 target address to be used. This is (or should be) actually the IPv6 address of the sender. The target address, if not specified using the -ta switch, is auto set to the IPv6 address of your machine.

-tm <TARGET_MAC>The MAC target address to be used. This is (or should be) actually the link-layer address of the sender. The target MAC (link-layer) address, if not specified using the -tm switch, is auto set to the MAC address of your machine.

-res <reserved> Reserved field. Default Value: 0

Neighbor Solicitation Messages

-neighsol Send neighbor advertisement messages. Default: False

-ta <TARGET_ADDRESS> The IPv6 target address to be used. This is (or should be) actually the IPv6 address of the target. The target address, if not specified using the -ta switch, is auto set to the IPv6 address of your machine.

-tm <TARGET_MAC>The MAC target address to be used. This is (or should be) the link-layer address of the sender. The target MAC (link-layer) address, if not specified using the *-tm* switch, is auto set to the MAC address of your machine.

-res <reserved> Reserved field. Default Value: 0

Redirect Message

- RFC 4861: They are sent to inform a host of a better first-hop node, or that the destination is in fact a neighbor.

Router Redirect Attacks

- Again, properly spoofed Router Redirect messages can be used to:
  - To put yourself in a middle for specific destination(s).
  - DoS (by putting a Fake Router)
  - DoS (by informing falsely that an off-link destination is on-link).

  etc...

Router Redirect

-rd Send Router Redirect (messages)

-da <DESTINATION_ADDRESS> The IPv6 destination address to be used in an ICMPv6 Router Redirect message

-ta <TARGET_ADDRESS> The IPv6 target address (Fake Router) to be used in an ICMPv6 Router Redirect message, or the same with the destination address if destination is a neighbor.

-tm <TARGET_MAC> The MAC target address (Fake Router) to be used in an ICMPv6 Router Redirect message

-rt <RANDOM_TARGET> Randomise the target IPv6 address to use as a Fake Router in an ICMPv6 Redirect message.

-pr <IPv6 prefix> The IPv6 network prefix to use. Example: fe80:224:54ff:feba:: Default="fe80::" This switch is used in combination with *-rt* switch.

Router Redirect Notes

- If *target_address* is not defined, your machine's source address is used (assuming that you want to place your machine as a router for the specific destination).
- If *destination_address* is not defined, “::” is used as a destination address.
- When use as a destination address the ff02::1 (multicast address), it auto-selects the broadcast destination MAC address 33:33:00:00:00:01.

  (this is also the case in all the other ND messages).

ICMPv6 Packet Too Big Messages

- Used to discover and take advantage of paths with PMTU greater than the IPv6 minimum link MTU.
- It makes possible two denial-of-service attacks, both based on a malicious party sending false Packet Too Big messages to a node.
  - In the first attack, the false message indicates a PMTU much smaller than reality. ... It will, however, result in suboptimal performance.
  - In the second attack, the false message indicates a PMTU larger than reality. This could cause <u>temporary</u> blockage as the victim sends packets that will be dropped by some router. ...Frequent repetition of this attack could cause lots of packets to be dropped.

ICMPv6 Packet Too Big Messages

-big	Send ICMPv6 Packet Too Big messages
-mtu <DMTU>	The MTU value to use.

IPv6 ND Messages Examples

Launch your sniffer (e.g. wireshark) to observe the packets that you craft, the traffic that you send and the responses.

Do not use wireshark filters this time.

Task 2

- Sent simple IPv6 Router Advertisement Multicast messages.
- Send Fake MTU information (bigger or smaller).
- DoS implicitly a router
  - No matter what the destination is
  - For specific destinations
- DoS specific off-link destinations
- Pretend yourself to be a router for specific destinations.
- Send spoofed Neighbor Advertisements messages.

IPv6 ND Messages Examples

- Simple IPv6 Router Advertisement Multicast messages

  ./chiron_nd.py vboxnet0 -ra -d ff02::1

- Fake MTU

  ./chiron_nd.py vboxnet0 -ra -mtu 3000 -d ff02::1

- Define Router Lifetime (in seconds)

  ./chiron_nd.py vboxnet0 -ra -mtu 3000 -m 07:00:00:00:00:01 -rl 0 -d ff02::1

IPv6 ND Messages Examples

- Set the Router priority to Low

  ./chiron_nd.py vboxnet0 -ra -mtu 3000 -m 07:00:00:00:00:01 -rl 0 -rp 3 -d ff02::1

- Advertise Specific IPv6 Network

  ./chiron_nd.py vboxnet0 -ra -mtu 3000 -m 07:00:00:00:00:01 -rl 0 -rp 1 -pr fe80:224:54ff:feba:: -pr-length 120 -d ff02::1

- Send Neighbor Advertisement messages

  ./chiron_nd.py vboxnet0 -neighadv -d fdf3:f0c0:2567:7fe4:a00:27ff:fe74:ddaa -tm 0a:00:27:00:00:01 -ta fdf3:f0c0:2567:7fe4:800:27ff:fe00:1 -r -o -sol

Experiment yourself

- How would you “mislead” IPv6 clients to use another IPv6 router?
- What kind of messages would you use?
- Experiment yourself :-)

3. An IPv4-to-IPv6 Proxy

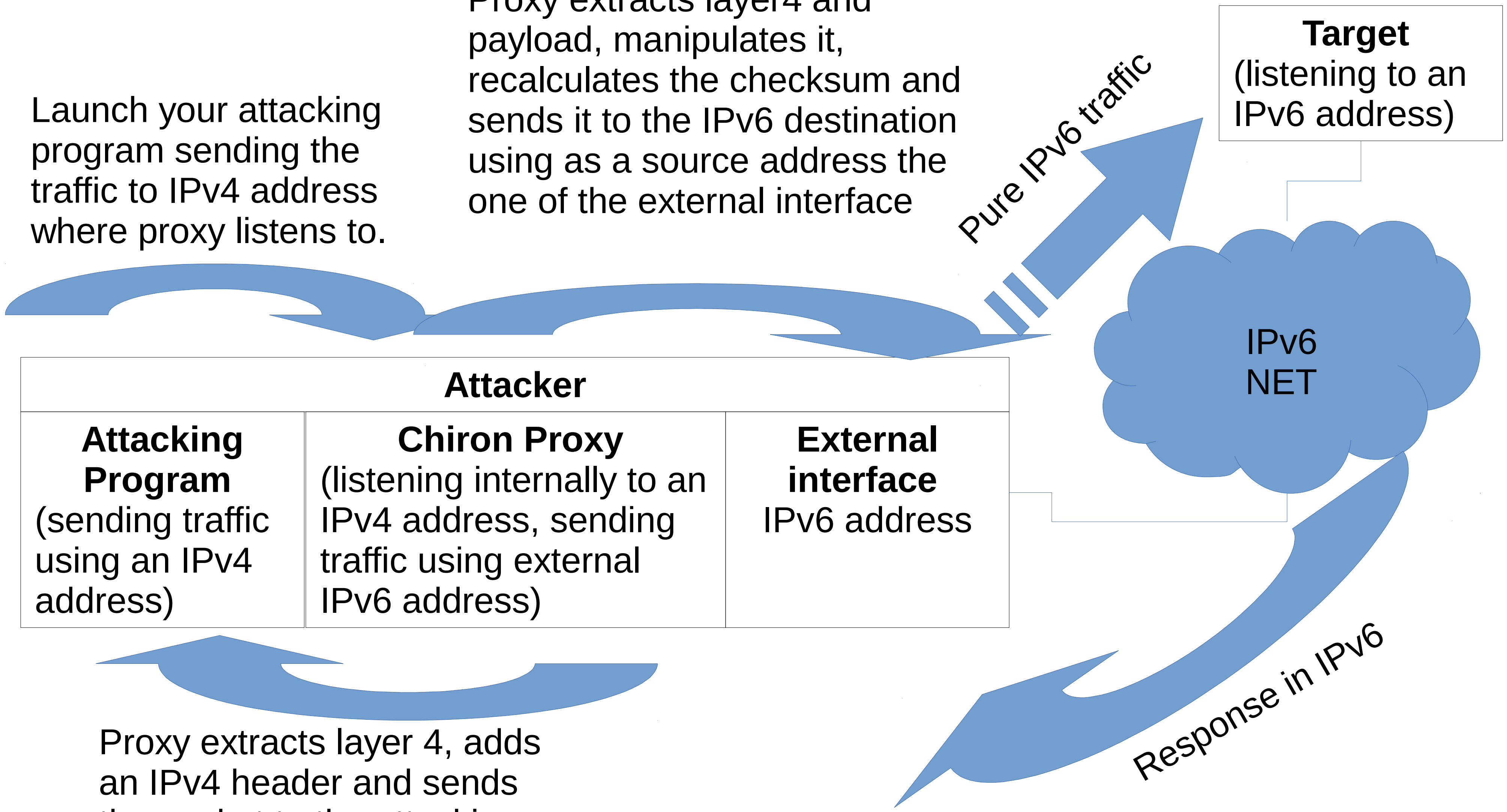
The Need for an IPv4-to-IPv6 Proxy

- 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 Connections


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

The Tricky Part
Configure the Local Firewall

- As already said, the framework does not use the OS stack but it's own library.
- When you send packets using the framework (e.g. a TCP SYN packet) and the other replies (SYN ACK in our example), your OS, which does not know anything about this, it will RESET (RST) the connection.
- To this end, you must temporarily configure your host firewall to drop such outgoing RST packets to the specific IPv6 destination.

Caution – Configure the Local Firewall

- For the time being, you have to do it on your own.
- In an updated version it will be configured automatically for you (at least for ip(6)tables and pf).

Task 3: An IPv4-to-IPv6 Proxy Example

- You need to launch nikto against an IPv6-enabled web server.
- Your target's IPv6 address is fdf3:f0c0:2567:7fe4:a00:27ff:fe74:ddaa
- Your machine's IPv6 address is fdf3:f0c0:2567:7fe4:800:27ff:fe00:0

IPv4-to-IPv6 Proxy – 1. Configure ip(6)tables

- ip6tables -I OUTPUT 1 -p icmpv6 --icmpv6-type destination-unreachable -s fdf3:f0c0:2567:7fe4:800:27ff:fe00:0 -d fdf3:f0c0:2567:7fe4:a00:27ff:fe74:ddaa -j DROP
- iptables -I OUTPUT 1 --source 127.0.0.3 --destination 127.0.0.1 -p tcp --tcp-flags RST RST -j DROP
- ip6tables -I OUTPUT 1 -p tcp --dport 80 -s fdf3:f0c0:2567:7fe4:800:27ff:fe00:0 -d fdf3:f0c0:2567:7fe4:a00:27ff:fe74:ddaa -j DROP

Target

Our host machine

IPv4-to-IPv6 Proxy Example
2. Launch the Proxy

```
./chiron_proxy.py vboxnet0 127.0.0.1 127.0.0.3
-d fdf3:f0c0:2567:7fe4:a00:27ff:fe74:ddaa
-threads 10
```

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.

IPv4-to-IPv6 Proxy Example
3. Run your Program

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

- It will take some extra time in comparison with direct communication, due to extra manipulation.

4. Manipulation Techniques of IPv6 Packets

They can be combined with the Scanner, the Proxy or the ND modules.

Manipulation Techniques of IPv6 Packets

- (Simple) fragmentation
- Flooding
- Crafting arbitrary IPv6 Extension Headers, regarding:
  - Type of Extension Headers
  - Number of occurrences of specific types of Extension
  - Order of Extension Headers
- All the above techniques can be combined with the Scanner, the Proxy or the ND modules.

Performing (Simple) Fragmentation

-nf <number_of_fragments>

-delay <number_of_fragments> sending delay between two consecutive fragments (in seconds).

How to Fragment Layer 4

- ***-l4_data <layer_4_data>*** the data (payload) of the layer4 protocol
- Examples:

./chiron_scanner.py vboxnet0 -d fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc -sn -l4_data "AAAAAAAA" ***-nf 2***

```
./chiron_scanner.py p10p1 -sn -d fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc -l4_data `python -c 'print "AABBCCDD" * 120'` -nf 4
```

In the last example, the layer-4 payload is 120 timed the “AABBCCDD” string.

Flooding Attacks

- Can be combined with all the pre described methods.

  -fl flood the targets

  -flooding-interval <FLOODING_INTERVAL> the interval between packets when flooding the targets (default: 0.1 seconds)

  -ftimeout <FLOODING_TIMEOUT>

  The time (in seconds) to flood your target (default: 200 seconds).

- Example:

  ```
  ./chiron_scanner.py p10p1 -d www.yahoo.com -rh0 -fl
  ```

Making Arbitrary IPv6 Extension Headers

or,

How to Fuzz (manually) IPv6 Protocol Implementation

Antonios Atlasis, aatlasis@secfu.net

IPv6 New Features: IPv6 Extension Headers

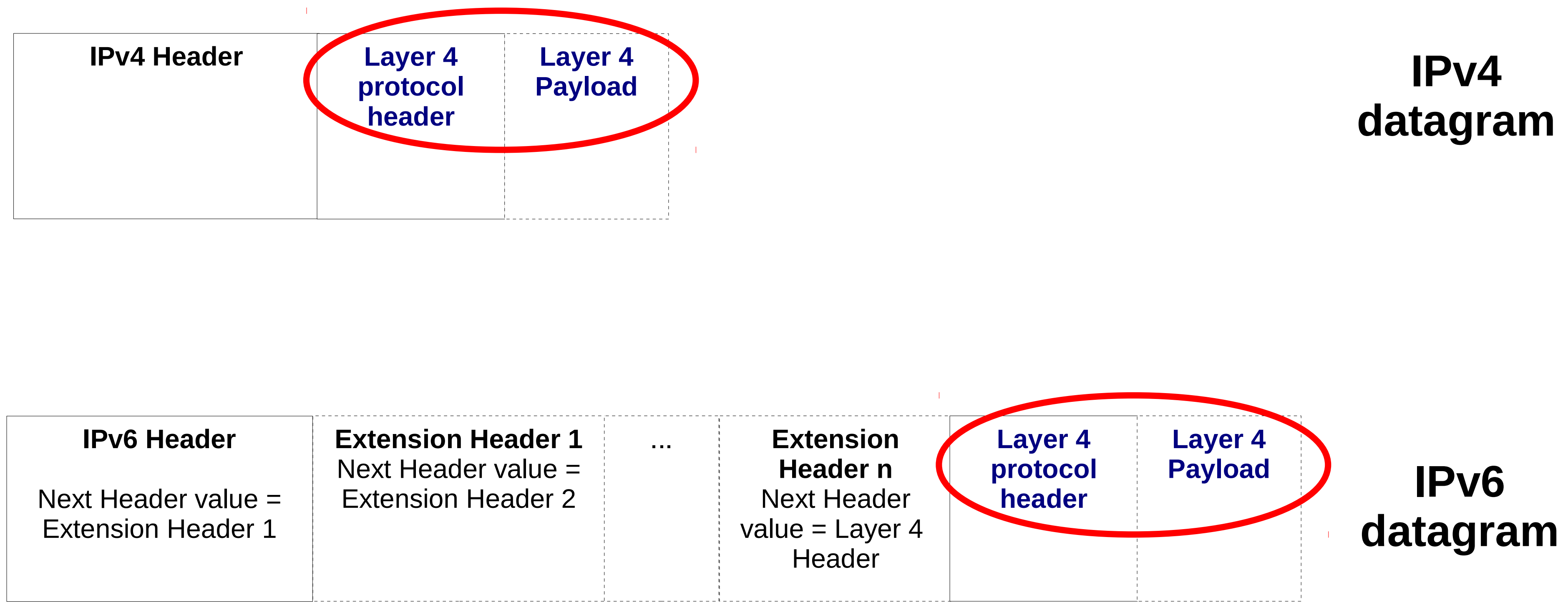
- It is not just the huge address space.
- One of the most significant changes: The introduction of the **IPv6 Extension Headers**.

The IPv6 Extension Headers

- 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.**

$\}$ Known from the IPSec

An IPv6 vs an IPv4 Datagram

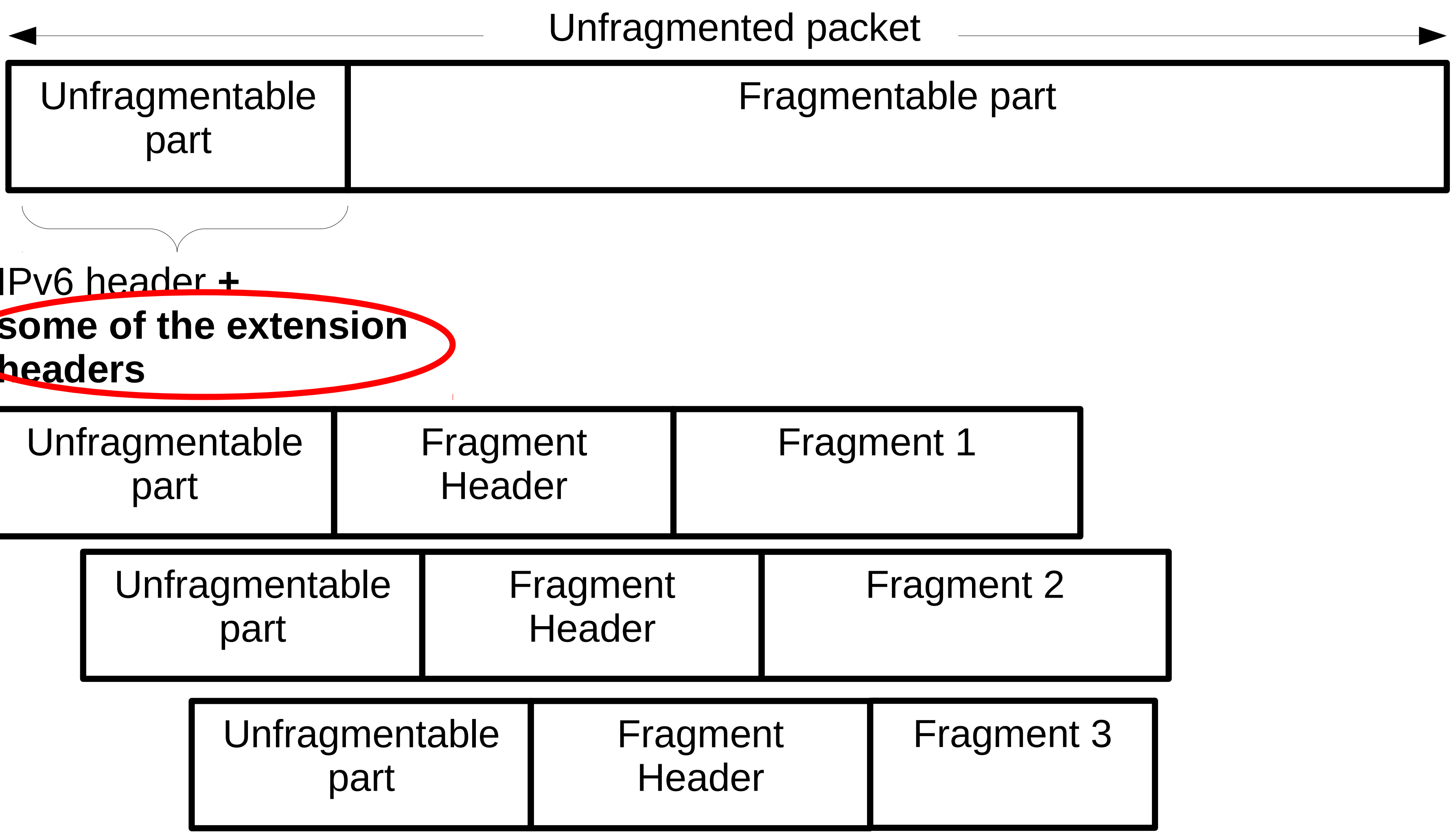


Fragmenting an IPv6 Header Chain

- *The **Unfragmentable Part** consists of the IPv6 header plus any extension headers <u>that must be processed by nodes en route </u>to the destination, that is, <u>all headers up to and including the Routing header if present</u>, else the Hop-by-Hop Options header if present, else no extension headers.*
- *The **Fragmentable Part** consists of the rest of the packet, that is, any extension headers that need be processed <u>only by the final destination node(s)</u>, plus the upper-layer header and data.*

Source: RFC 2460

IPv6 Fragmentation



Fuzzing (Manually) IPv6 Extension Headers

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

-luE <comma_separated_list_of_headers_that_remain_unfragmented>
Define 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:

Token 3: 9457-1932-4132-3902-3458

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), *reserved* (the reserved field), *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), *res1* (1st reserved field),*res2* (2nd reserved field)
60	Destination Options Header	optdata, otype

IPv6 Extension Headers Examples

Antonios Atlasis, aatlasis@secfu.net

Task 4: Adding Several IPv6 Extension Headers

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

  ```
  ./chiron_scanner.py vboxnet0 -d fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc -sn -luE 60
  ```

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

  ```
  ./chiron_scanner.py vboxnet0 -d fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc -sn -luE 0,60
  ```

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

  ```
  ./chiron_scanner.py vboxnet0 -d fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc -sn -luE 0,3x60
  ```

Fragment Layer 4 and Some of the IPv6 Extension Headers

```
./chiron_scanner.py vboxnet0 -d
fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc -sn -luE
0,3x60 -lfE 2x60 -l4_data "AAAAAAAA" -nf 4
```

Increasing the Size of the Options Header Arbitrarily

-seh <SIZE_OF_EXTHEADERS> the size of the Options Extension header in octets of bytes.

Example:

```
./chiron_scanner.py vboxnet0 -d
fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc -sn -lfE 60 -nf 4 -seh 3
```

Define Explicitly the Values of the IPv6 Extension Headers

44"(offset=3;res1=3;m=1;res2=234)"

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

  ./chiron_scanner.py vboxnet0 -d fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc -luE 0"(otype=128;optdata=AAAAAAA)" -sn

- **Destination Options Header**

  ./chiron_scanner.py vboxnet0 -d fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc -luE 60"(otype=128;optdata=AAAAAAAA)" -sn

- **Type 0 Routing Header**

  ./chiron_scanner.py vboxnet0 -d fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc -luE 43"(type=0;addresses=2002::1-2002::2;segleft=2)" -sn

Defining Explicitly the Values of the IPv6 Extension Headers

- **IPv4 Tunneling**

```
./chiron_scanner.py vboxnet0 -d fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc -luE 4"(src=192.156.55.44;dst=38.55.44.3)" -sn
```

- **IPv4 Tunneling preceded by a Destination Options Header**

```
./chiron_scanner.py vboxnet0 -d fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc -luE 60,4"(src=192.156.55.44;dst=38.55.44.3)" -sn
```

- **IPv6 Tunneling preceded by two Destination Options Header and three Fragment Extension Headers**

```
./chiron_scanner.py vboxnet0 -d fd9e:488f:c9e9:b6fd:a00:27ff:fe10:8fc -luE 2x60"(otype=128;optdata=AAAAAAAA)",3x44"(offset=3;res1=3;m=1;res2=234)" -sn
```

“Playing” With The Next Header Values of the IPv6 Ext. Headers

Antonios Atlasis, aatlasis@secfu.net

Each Fragment is Composed Of

- The Unfragmentable Part of the original packet,...and the Next Header field of the last header of the Unfragmentable Part changed to 44.
- A Fragment header containing:
  - The Next Header value that identifies the first header of the Fragmentable Part <u>of the original</u> packet.

Source: RFC 2460

Reassembling a Fragmented IPv6 Datagram

- *The **Unfragmentable Part** of the reassembled packet consists of all headers up to, but not including, the Fragment header of the first fragment packet (that is, the packet whose Fragment Offset is zero), with the following change(s):*
  - *The **Next Header** field of the last header of the Unfragmentable Part is obtained from the Next Header field <u>of the first fragment's Fragment header</u>.*

Source: RFC 2460

Abusing The Next Header Values Using Chiron

- You can change the Next Header value of the Fragment Header of the last fragment to the Layer-4 header value (instead of the correct header value, which should be the one of the 1st Extension Header of the Fragmentable part), using the ***-wc*** switch.

Abusing The Next Header Values: Example

- 1st Fragment:

  IPv6 main Header + Fragment Ext Header (offset =0, M=1, next header =60) + Dest Opt Header (8 bytes long, no data on it but padding, next header = 6)

- 2nd Fragment:

  IPv6 main header + Fragment Ext Header (offset=1, M=0, next header = 6) + TCP header.

The Good Stuff is,

- That you can combine all the IPv6 Extension Headers techniques with:
  - Network scanning
  - On the local-link using the corresponding ND messages.
  - With the proxy
  - With fragmentation,
  - With flooding, etc

Example: Evasion Techniques

- Use 8 to 10 Fragment Extension Headers in an Atomic Fragment
- Fragment the IPv6 datagram and send the layer-4 header at fragment 10 or later.
- Construct the IPv6 header chains using wrong next header values: Use the ***-wc*** switch.

Future Work

- IPv6 Attacking module
  - Will implement some of the most well-known IPv6 attacking tool.
  - Although already implemented by other frameworks, it will be possible to combine them with arbitrary IPv6 Extension Headers.
- Auto-Fuzzing of the IPv6 Extension Header parameters.
- Fragmentation overlapping
- Multi-processing instead of multi-threading.
- The Chiron project is also supported by the Brucon 5x5 program.
- Updated versions will be uploaded at www.secfu.net.

Please, Keep in Touch

- For:
  - bugs (there should be many),
  - new requested features / modules / capabilities
  - Any other comments / proposals, etc.
- You can reach me at: aatlasis@secfu.net

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?

Wait: We haven't finished yet :-)

Challenge

- Find a technique to evade Snort.
  - Use simple ping request first
- Then, try chiron proxy and nikto to attack a web server.
- By aware, Snort is watching you!