# Securing IPv6 Networks: ft6 & friends

Oliver Eggert, Simon Kiertscher

# Our Group

# Outline

- IPv6 Intrusion Detection System Project
- IPv6 Basics
- Firewall Tests
- FT6 (Firewall test tool for IPv6)

# IPv6 Intrusion Detection System

- Partners:
  - University of Potsdam
  - Beuth University of Applied Sciences Berlin
  - EANTC AG
- Associated Partner:
  - STRATO AG
- Funded by the Federal Ministry of Education and Research

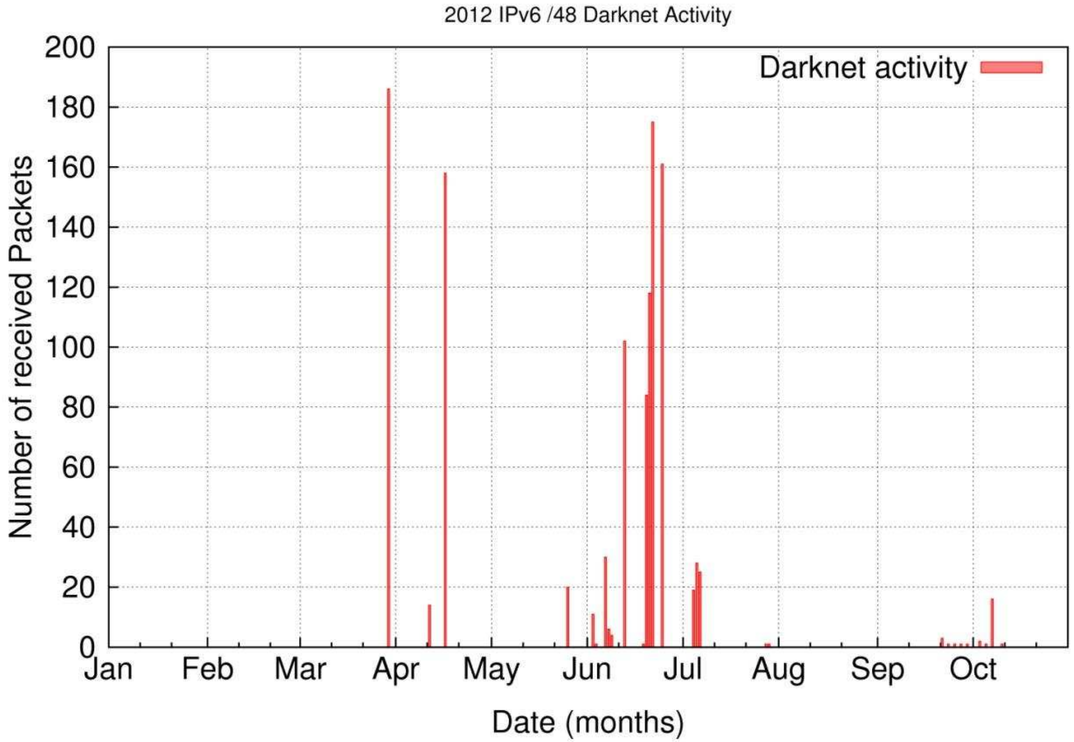
# IPv6 Intrusion Detection System

## Main contributions of the project

1. Test operation of an IPv6 Darknet
2. Honeyd → Honeydv6
3. Snort IPv6-Plugin (IDS/IPS Software)
4. Load tests
5. Protocol tests

# Test operation of a Darknet

- /48 net, after 9 months 1172 packets captured
- Probably only backscatter traffic



## Honeyd → Honeydv6

- first low-interaction honeypot which can simulate entire IPv6 networks on a single host
- based on open source low-interaction honeypot honeyd developed by Niels Provos
- custom network stack to simulate thousands of hosts
- new protocols like NDP and ICMPv6 implemented
- updated routing engine to simulate entire network topologies
- extension header processing implemented
- observe fragmentation based IPv6 attacks
- source code available on www.idsv6.de

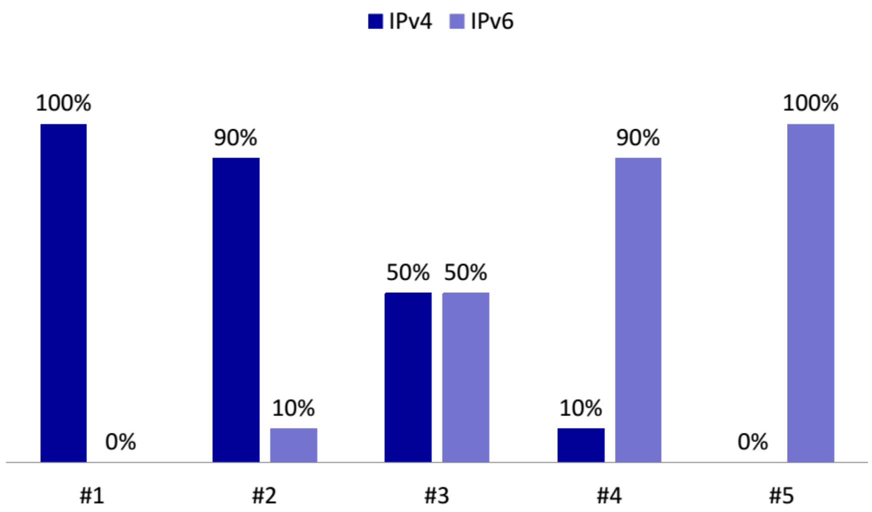
# Snort IPv6-Plugin

- Widely used Open Source NIDS
- Snort IPv6 support technically yes, but . . .

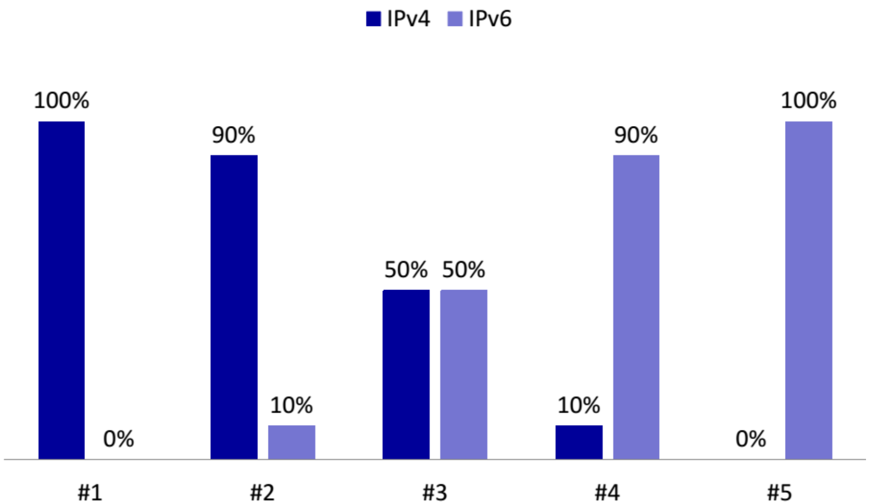
- Snort IPv6 Plugin (Preprocessor)
- Functionality:
  - Reads ICMPv6 messages on the LAN
  - Follows network state, i. e. (MAC, IP) of:
    - On-link Routers
    - On-link Hosts
    - Ongoing Duplicate Address Detection
  - Alerts on new/unknown hosts and routers

- All IPv6 fields accessible for Snort signatures now
  - Basic Header, Extension Headers, Neighbor Discovery Options

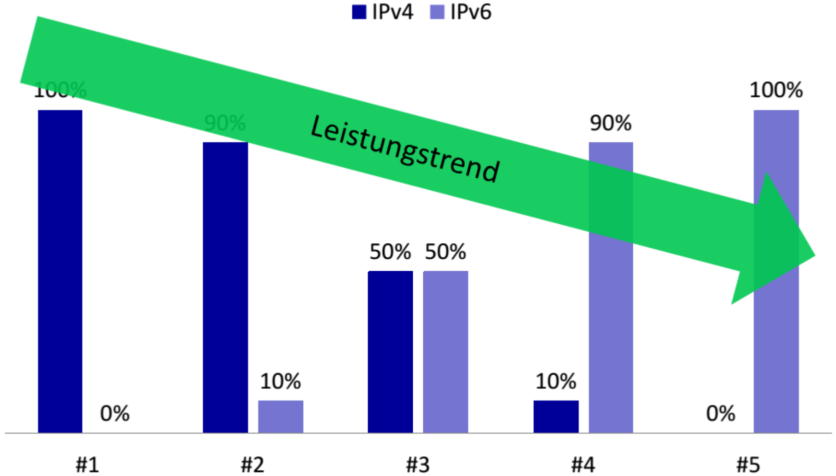
# Load tests



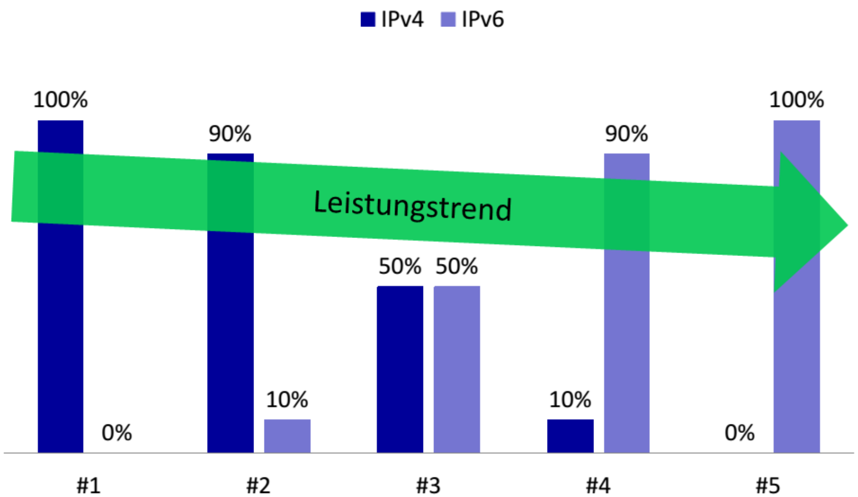
# Load tests



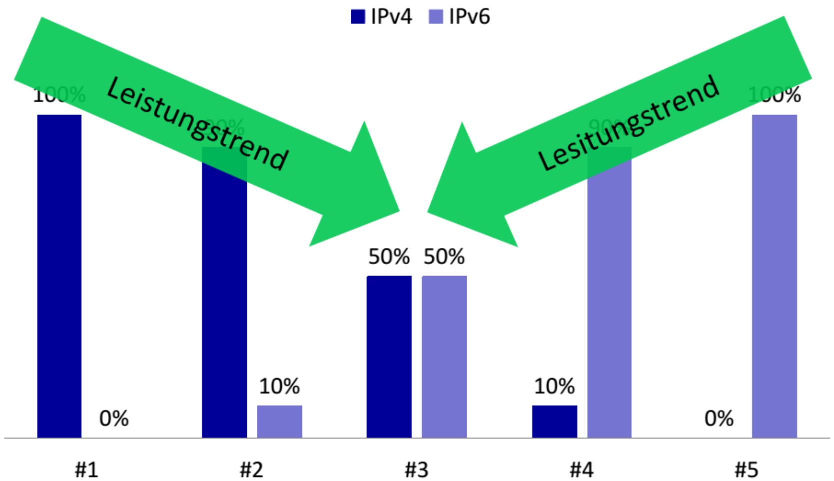
IPv4 IPv6
100%
0%
90%
10%
50% 50%
10%
90%
0%
100%
#1
#2
#3
#4
#5

IPv4 IPv6
Leistungstrend
0%
10%
50% 50%
10%
90%
0%
100%
#1
#2
#3
#4
#5

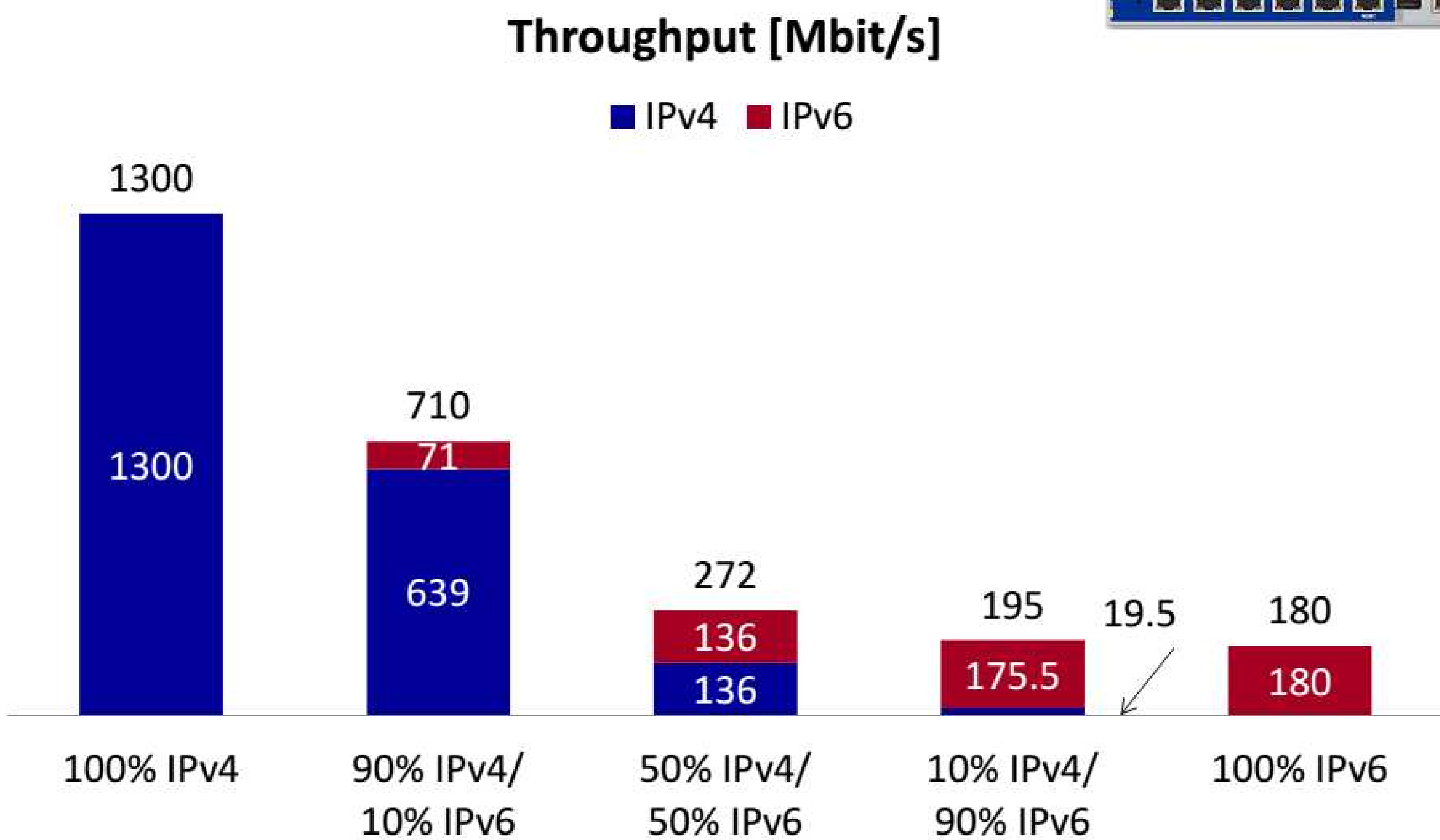
IPv4 IPv6
100%
90%
Leistungstrend
10%
50% 50%
10%
90%
0%
0%
100%
#1
#2
#3
#4
#5

IPv4 IPv6
Leistungstrend
Lesitungstrend
50% 50%
0%
10%
10%
0%
#1
#2
#3
#4
#5

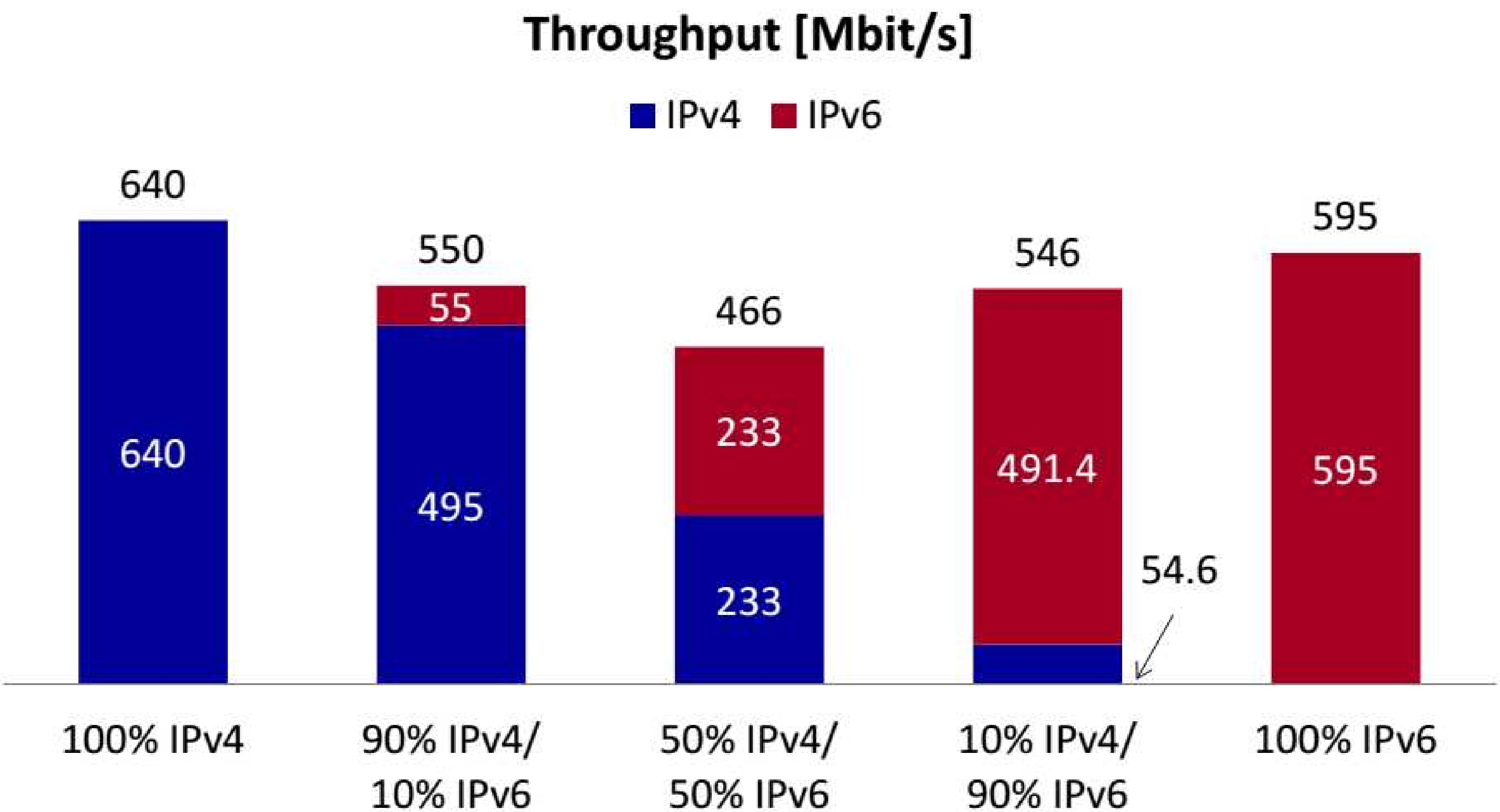
# Load tests


Check Point


Throughput [Mbit/s]
IPv4
IPv6
1300
1300
710
71
639
272
136
136
195
175.5
19.5
180
180
100% IPv4
90% IPv4/ 10% IPv6
50% IPv4/ 50% IPv6
10% IPv4/ 90% IPv6
100% IPv6

# Load tests


Juniper
J2320

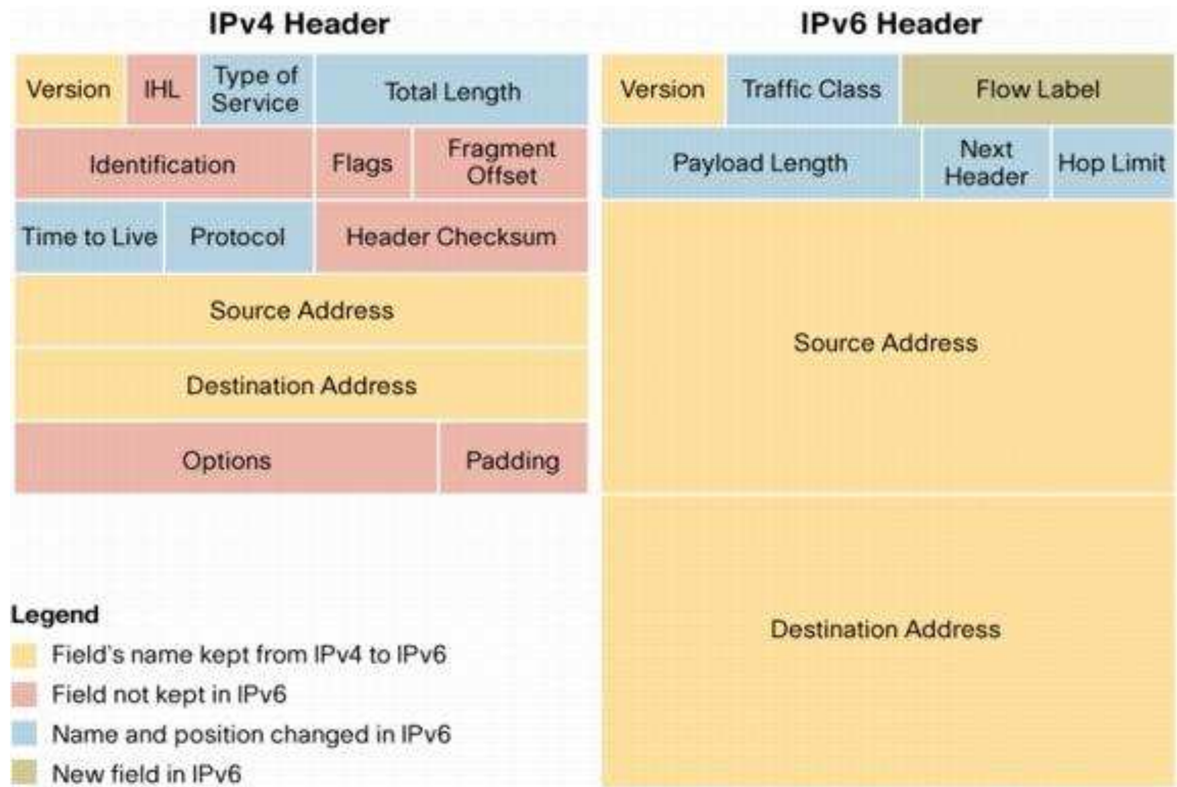

Throughput [Mbit/s]
IPv4
IPv6
640
640
550
55
495
466
233
233
546
491.4
54.6
595
595
100% IPv4
90% IPv4/ 10% IPv6
50% IPv4/ 50% IPv6
10% IPv4/ 90% IPv6
100% IPv6

# IPv6 Basics

# IPv6 Basics

- IPv4
  Optional options and padding → Variable header size

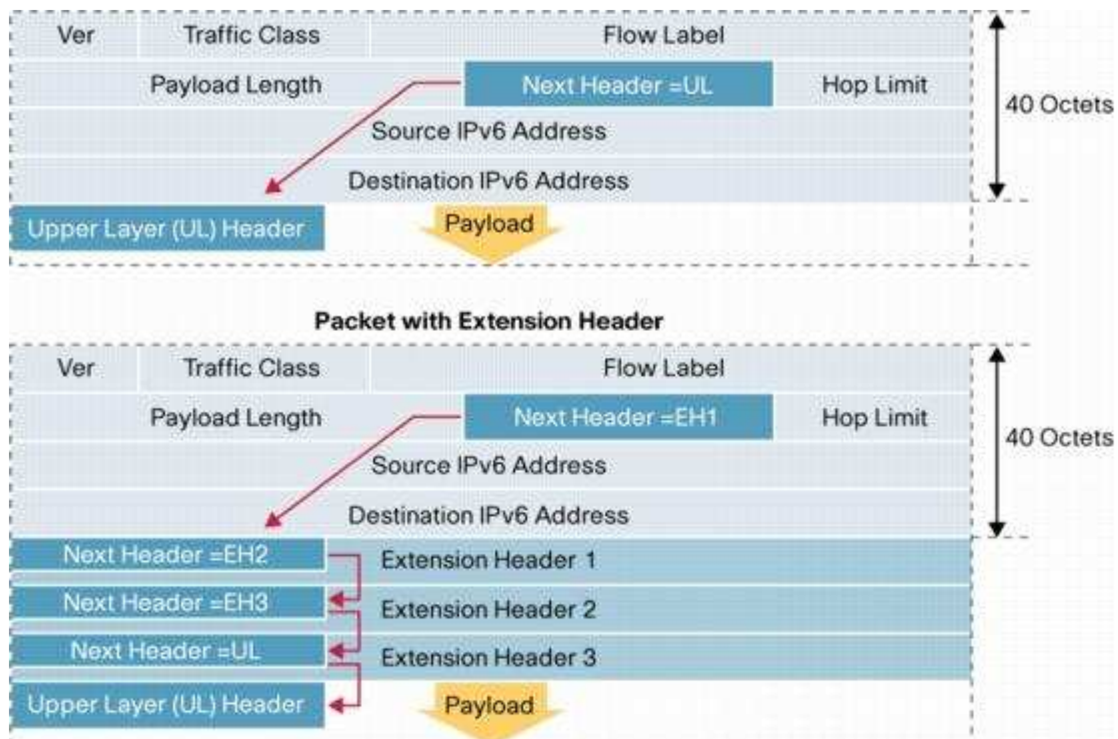
- IPv6
  Fixed but bigger header size
- Options?
  → extension headers




Source:
http://www.cisco.com/en/US/technologies/tk648/tk872/images/technologies_white_paper0900aecd8054d37d-03.jpg

# IPv6 Basics - Extension Headers

- Hop-By-Hop Options
- Routing Header
- Fragment Header
- Authentication Header
- Encapsulating Security Payload
- Destination Options
- Mobility Header
- No Next Header




Source: http://www.cisco.com/en/US/technologies/tk648/tk872/images/technologies_white_paper0900aecd8054d37d-04.jpg

# Firewall Tests

# Motivation

- What are the RFC requirements for IPv6 firewalls?
- How can you test your firewall in an easy way?
- Can “IPv6 Ready” hardware keep its promise?

# ICMPv6 filtering

- ICMPv6 is like ICMP for sharing information or error messages
- BUT:
  New ICMPv6 types for Neighbor Discovery Protocol (NDP, the former ARP) and Multicast Listener Discovery Protocol (MLD)
- Do not drop all ICMPv6 messages mindlessly

# ICMPv6 filtering

- Non-Filtered messages according to RFC 4890

| ICMPv6 Type | Description |
|---|---|
| 1 | Destination Unreachable |
| 2 | Packet Too Big |
| 3, Code 0 | Time Exceeded |
| 4, Code 1 and 2 | Parameter Problem |
| 128, 129 | Echo Request/Reply |

# ICMPv6 filtering

- Optional Filter List

| ICMPv6 Type | Description |
|---|---|
| 3, Code 1 | Time Exceeded |
| 4, Code 0 | Parameter Problem |
| 144, 145, 146, 147 | IPv6 Mobility |
| 150 | Seamoby Experimental |
| 5-99, 102-126 | Unallocated Error Messages |
| 154-199, 202-254 | Unallocated Informational Messages |

- The rest should be filtered!

# Routing Header (RH)

- Especially RH0 (deprecated since Dec 2007 according to RFC 5095)

  → treat it like an unknown RH

- Mobility Routing Header (RH 2) - RFC 3775

| RH Type | Segments left field | Behavior |
|---|---|---|
| RH 0 | ≠ 0 | Drop |
| RH 0 | = 0 | Forward (ignore header) |
| RH 2 | ≠ 1 | Drop |
| RH 2 | = 1 | Forward |
| RH 200 | ≠ 0 | Drop |
| RH 200 | = 0 | Forward (ignore header) |

# IPv6 Header Chain Inspection

There are 3 basic rules (RFC2460) that govern the order and occurrence of extension headers (header chain)

1. Destination Options (DSTOPT) header at most twice (once before a Routing header and once before the upper-layer header)
2. All other extension headers should occur at most once
3. The Hop-by-Hop (HBH) Options header is restricted to appear only immediately after the base IPv6 header

# IPv6 Header Chain Inspection

We test 7 different Header Chains

| Header Chain | Validity |
|---|---|
| DSTOPT | Valid |
| DSTOPT, DSTOPT | Invalid |
| DSTOPT, RH, DSTOPT | Valid |
| HBH | Valid |
| HBH, HBH | Invalid |
| DSTOPT, HBH | Invalid |
| HBH, DSTOPT, RH, HBH | Invalid |

# Overlapping IPv6 Fragments

RFC 5722 “Handling of Overlapping IPv6 Fragments” describes e.g. a fragmentation attack and expected node behavior

| Fragment appearance | Behavior |
|---|---|
| Fragmented packet without overlap | Forward |
| Overlapping, rewriting the upper layer protocol header | Drop |
| Overlapping, rewriting the payload | Drop |

# Overlapping IPv6 Fragments

```
372 25.285318 2001:2:1::b 2001:2:2::b IPv6   IPv6 fragment (nxt=UDP (17) off=0 id=0x532fbc21)
373 25.349511 2001:2:1::b 2001:2:2::b UDP    Source port: krb524  Destination port: ssh
374 25.428852 2001:2:1::b 2001:2:2::b IPv6   IPv6 fragment (nxt=UDP (17) off=0 id=0x21c24a47)
375 25.490046 2001:2:1::b 2001:2:2::b UDP    Source port: krb524  Destination port: http
376 25.523564 2001:2:1::b 2001:2:2::b TCP    [TCP segment of a reassembled PDU]
379 25.524289 2001:2:1::b 2001:2:2::b TCP    39296 > http [ACK] Seq=81 Ack=27037 Win=62976 Len=0 TSval=154793 TSecr=127430
381 25.525069 2001:2:1::b 2001:2:2::b TCP    39296 > http [ACK] Seq=81 Ack=27050 Win=62976 Len=0 TSval=154793 TSecr=127430
383 26.526692 2001:2:1::b 2001:2:2::b TCP    39296 > http [ACK] Seq=81 Ack=27122 Win=62976 Len=0 TSval=155043 TSecr=127681
385 26.527111 2001:2:1::b 2001:2:2::b TCP    39296 > http [ACK] Seq=81 Ack=27288 Win=64512 Len=0 TSval=155043 TSecr=127681
386 26.527275 2001:2:1::b 2001:2:2::b TCP    [TCP segment of a reassembled PDU]
```

```
⊟ Internet Protocol Version 6, Src: 2001:2:1::b (2001:2:1::b), Dst: 2001:2:2::b (2001:2:2::b)
  ⊞ 0110 .... = Version: 6
  ⊞ .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 160
    Next header: IPv6 fragment (44)
    Hop limit: 64
    Source: 2001:2:1::b (2001:2:1::b)
    Destination: 2001:2:2::b (2001:2:2::b)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
  ⊟ Fragmentation Header
      Next header: UDP (17)
      Reserved octet: 0x0000
      0000 0000 0000 0... = Offset: 0 (0x0000)
      .... .... .... .00. = Reserved bits: 0 (0x0000)
      .... .... .... ...1 = More Fragment: Yes
      Identification: 0x532fbc21
⊟ Data (152 bytes)
    Data: 115c005000986acd61616161616161616161616161616161...
    [Length: 152]
```

**0x50 = 80 = http**

```
0000  00 10 18 4f a9 48 18 03  73 c1 e7 3c 86 dd 60 00   ...O.H.. s..<..`.
0010  00 00 00 a0 2c 40 20 01  00 02 00 01 00 00 00 00   ....,@ . ........
0020  00 00 00 00 00 0b 20 01  00 02 00 02 00 00 00 00   ...... . ........
0030  00 00 00 00 00 0b 11 00  00 01 53 2f bc 21 11 5c   ........ ..S/.!.\
0040  00 50 00 98 6a cd 61 61  61 61 61 61 61 61 61 61   .P..j.aa aaaaaaaa
0050  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
0060  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
0070  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
0080  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
0090  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
00a0  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
00b0  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
00c0  61 61 61 61 61 61 58 58  58 58 58 58 54 65 73 74   aaaaaaXX XXXXTest
00d0  34 53 74 65 70 32                                  4Step2
```

# Overlapping IPv6 Fragments

```
372 25.285318 2001:2:1::b  2001:2:2::b IPv6    IPv6 fragment (nxt=UDP (17) off=0 id=0x532fbc21)
373 25.349511 2001:2:1::b  2001:2:2::b UDP     Source port: krb524  Destination port: ssh
374 25.428852 2001:2:1::b  2001:2:2::b IPv6    IPv6 fragment (nxt=UDP (17) off=0 id=0x21c24a47)
375 25.490046 2001:2:1::b  2001:2:2::b UDP     Source port: krb524  Destination port: http
376 25.523564 2001:2:1::b  2001:2:2::b TCP     [TCP segment of a reassembled PDU]
379 25.524289 2001:2:1::b  2001:2:2::b TCP     39296 > http [ACK] Seq=81 Ack=27037 Win=62976 Len=0 TSval=154793 TSecr=127430
381 25.525069 2001:2:1::b  2001:2:2::b TCP     39296 > http [ACK] Seq=81 Ack=27050 Win=62976 Len=0 TSval=154793 TSecr=127430
383 26.526692 2001:2:1::b  2001:2:2::b TCP     39296 > http [ACK] Seq=81 Ack=27122 Win=62976 Len=0 TSval=155043 TSecr=127681
385 26.527111 2001:2:1::b  2001:2:2::b TCP     39296 > http [ACK] Seq=81 Ack=27288 Win=64512 Len=0 TSval=155043 TSecr=127681
386 26.527275 2001:2:1::b  2001:2:2::b TCP     [TCP segment of a reassembled PDU]
```

```
    Next header: IPv6 fragment (44)
    Hop limit: 64
    Source: 2001:2:1::b (2001:2:1::b)
    Destination: 2001:2:2::b (2001:2:2::b)
    [Source GeoIP: Unknown]
    [Destination GeoIP: Unknown]
  ⊟ Fragmentation Header
      Next header: UDP (17)
      Reserved octet: 0x0000
      0000 0000 0000 0... = Offset: 0 (0x0000)
      .... .... .... .00. = Reserved bits: 0 (0x0000)
      .... .... .... ...0 = More Fragment: No
      Identification: 0x532fbc21
⊟ User Datagram Protocol, Src Port: krb524 (4444), Dst Port: ssh (22)
    Source port: krb524 (4444)
    Destination port: ssh (22)
    Length: 152
  ⊞ Checksum: 0x6b07 [validation disabled]
⊟ Data (144 bytes)
    Data: 6161616161616161616161616161616161616161616161616161...
    [Length: 144]
```

```
0000  00 10 18 4f a9 48 18 03  73 c1 e7 3c 86 dd 60 00   ...O.H.. s..<..`.
0010  00 00 00 a0 2c 40 20 01  00 02 00 01 00 00 00 00   ....,@ . ........
0020  00 00 00 00 00 0b 20 01  00 02 00 02 00 00 00 00   ...... . ........
0030  00 00 00 00 00 0b 11 00  00 00 53 2f bc 21 11 5c   ........ ..S/.!.\
0040  00 16 00 98 6b 07 61 61  61 61 61 61 61 61 61 61   ....k.aa aaaaaaaa
0050  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
0060  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
0070  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
0080  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
0090  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
00a0  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
00b0  61 61 61 61 61 61 61 61  61 61 61 61 61 61 61 61   aaaaaaaa aaaaaaaa
00c0  61 61 61 61 61 61 58 58  58 58 58 58 54 65 73 74   aaaaaaXX XXXXTest
00d0  34 53 74 65 70 32                                  4Step2
```

# Tiny IPv6 Fragments

- A Tiny-Fragment is a fragmented IPv6 packet where the upper-layer-header is located in the second fragment
- Firewall has to inspect the second fragment

| Tiny Fragment appearance | Behavior |
| --- | --- |
| Upper-layer-header with allowed port number | Forward |
| Upper-layer-header with forbidden port number | Drop |

# Tiny IPv6 Fragments

According RFC 2460 a device has to discard a packed if not all fragments have arrived within 60 seconds after the arrival of the first fragment

| Tiny Fragment appearance | Behavior |
|---|---|
| Send the last fragment after 60 seconds | Forward |
| Send the last fragment after 61 seconds | Drop |

# Excessive Hop-by-Hop and Destination Option Options

- Excessive use → denial-of-service attack
- As specified in RFC 4942, every option should occur at most once, except Pad1 and PadN
- All HBH options have to be processed on every node they pass

| Options Profile |
|---|
| Jumbo Payload, PadN, Jumbo Payload |
| Router Alert, Pad1, Router Alert |
| Quick Start, Tunnel Encapsulation Limit, PadN, Quick Start |
| RPL Option, PadN, RPL Option |

# PadN Covert Channel

- PadN and Pad1 are used to align options to a multiple of 8 bytes
- Required for DSTOPT and HBH header
- Valid payload of PadN must only contains zeroes

→ Abuse as a covert channel

| Header | PadN | Behavior |
| --- | --- | --- |
| HBH | Valid | Forward |
| HBH | Invalid | Drop |
| DSTOPT | Valid | Forward |
| DSTOPT | Invalid | Drop |

# Address Scopes

- A firewall must not forward packets with a wrong scope address
- The test contains a mix of different
  - Multicast addresses
  - Link-local addresses

| Scope | Address range |
|---|---|
| Multicast | ff00::/32 - ffff::/32 |
| Link-Local | fe80::/16 - febf::/16 |

# FT6
# Technical Stuff

# ft6 – Motivation

- next step: perform the tests
- usually tedious, error prone work
- aided by a tool
- easily reproducable, comparable
- enter ft6

# ft6 – Agenda

1. overview
2. info on design and implementation
3. live demo
4. v.2: security focus
5. writing your own tests *(optionally)*

# ft6 – Design Goals

- easy to configure
- graphical user interface
- browse tests and results
- visual representation

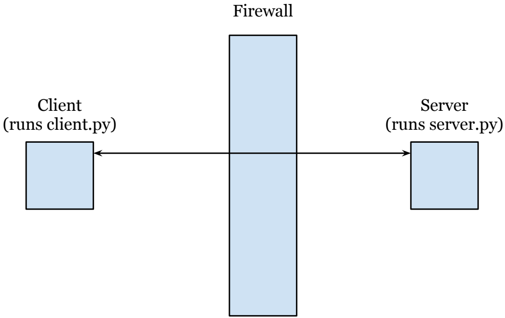
# ft6 – Design Goals

- open-source (Creative Commons BY-NC-SA 3.0)
- can act as a framework for new tests
- easy to implement new tests

# ft6 – Details

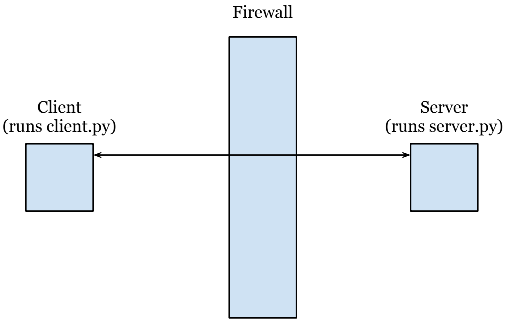
- powered by python, PyQt and scapy
- works with Linux, Windows 7, OS X
- python: rapid developement, easily understandable
- PyQt: GUI-framework, available cross-platform
  - http://www.riverbankcomputing.com/software/pyqt/intro
- scapy: great framework for network packet creation
  - http://www.secdev.org/projects/scapy/

# ft6 – Architecture



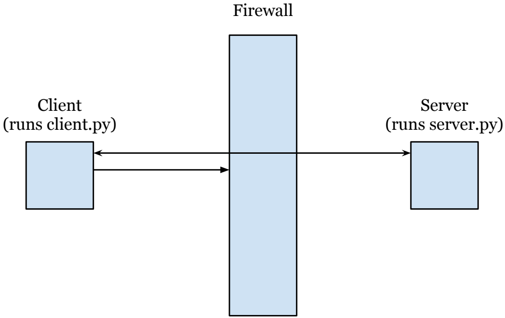

- ft6 is a client-server application
- requires machines on both sides of your firewall
- one open port
- place machines not more than one hop away from firewall

# ft6 – Running ft6



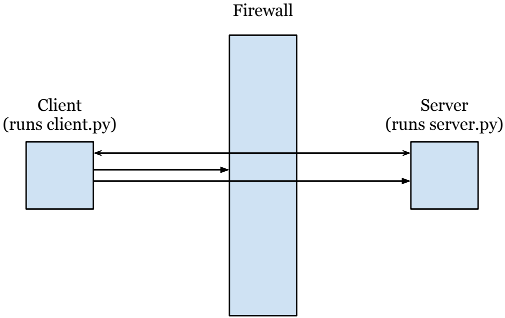

- Client and Server exhange control messages
  - Start / End / Results

# ft6 – Running ft6



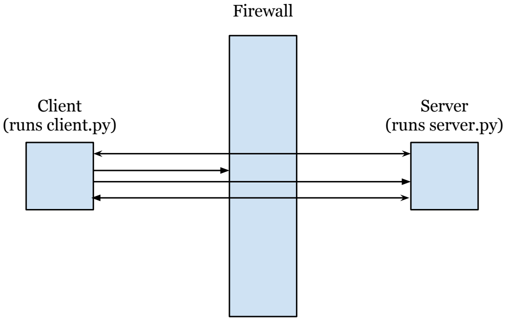

- Client sends packets
- Server sniffs

# ft6 – Running ft6




- Client sends packets
- Server sniffs

# ft6 – Running ft6




- Server sends back list of packets it recieved
- Client figures out what went missing and displays result

Live Demo

# ft6 version 2: pitfalls

- ideal world scenario: tests performed automatically
- mismatch between rfc's intent, your setup, firewall capabilities
- ft6's results may be misleading in some cases

# ft6 version 2: pitfalls

Example:

- ICMPv6 non-filtered messages include `Packet Too Big`, `Time Exceeded` and `Parameter Problem`
- in our tests: were dropped by some firewalls, marked red in ft6
- responses to some previous malformed packet
- ft6 doesn't send the previous packet
- firewall more capable than assumed

# ft6 version 2: pitfalls

- how would you test that?
- you can't (reliably)
- too many edge-cases, to many differences across vendors
- problem remains: what's the result of that ICMP test?

# ft6 version 2: pitfalls

another example: Routing Header

- decision to drop or forward depends upon value of `segments-left` field.
- some firewalls were unable to inspect the field.
- all or nothing
- firewall less capable than assumed
- yet: dropping valid RH is arguably better than forwarding invalid RH
- how do we reflect that in ft6?

# ft6 version 2: "security focus"

- switch from *rfc-conformity* focus to *security* focus
- if a result is not in accordance with rfc but "more secure":
  $\Rightarrow$ no longer red
- can't make it green:
  $\Rightarrow$ for example: dropping *all* RH, kills Mobile-IPv6 feature

# ft6 version 2: "security focus"

results:

- more yellow, longer explanations
- more interpretation required
- shows problems of IPv6. Too many *what-if*s

# ft6 – future work

- ft6 is a work in progress
- lots of improvement could be done
- better results
- more tests

# Thank You! Questions?

- your thoughts: contact@idsv6.de
- get ft6 from: https://redmine.cs.uni-potsdam.de/projects/ft6
- more info on the project: www.idsv6.de
- article in c't: www.ct.de/inhalt/2013/15/36

# ft6 – Writing your own test

Example: build own test, to see if packets containing the string "randomword" can traverse the firewall. Requires four steps:

1. create a class for your test
2. implement the `execute` method
3. implement the `evaluate` method
4. register your test with the application

(More detailed in ft6's documentation)

# ft6 – Writing your own tests

## Step 1: Create a class for your test

```
class TestRandomWord(Test):
  def __init__(self, id, name, description, test_settings, app):
    super(TestRandomWord, self).__init__(id, name, description,
      test_settings, app)
```

# ft6 – Writing your own tests

## Step 2: implement the `execute` method

```
def execute(self):
  e  = Ether(dst=self.test_settings.router_mac)
  ip = IPv6(dst=self.test_settings.dst, src=self.test_settings.src)
  udp= UDP(dport=self.test_settings.open_port, sport=12345)
  payload = "ipv6-qab"*128

  packet = e/ip/udp/(payload + "randomword")
  sendp(packet)

  packet = e/ip/udp(payload + "someotherword")
  sendp(packet)
```

# ft6 – Writing your own tests

## Step 3: implement the `evaluate` method

```
def evaluate(self, packets):
  results = []
  found_random = False
  found_otherword = False

   # iterate over the packets, filter those that belong to the test
   for p in packets:
      tag = str(p.lastlayer())
      if not "ipv6-qab" in tag:
          continue

      if "randomword" in tag:
          found_random = True

      if "someotherword" in tag:
          found_otherword = True
```

# ft6 – Writing your own tests

## Step 3: implement the `evaluate` method

```
# evaluate the flags
if found_random:
      results.append("Success", "Your firewall forwarded
      a packet with a random word!")
else:
      results.append("Failure", "Your firewall dropped
      a packet with a random word!")

if found_otherword:
    results.append("Warning", "Your firewall forwarded
    a packet with some other word. That's very weird!")
else:
    results.append("Success", "Your firewall dropped
    a packet with some other word. Well done firewall!")

return results
```

# ft6 – Writing your own tests

## Step 4: register your test

```
# create test classes, store them in the dictionary
# so they can later be called by their id
tICMP = TestICMP(1, "ICMPv6 Filtering", "The ICMP Test",
  self.test_settings, app)
...
tRandomWord = TestRandomWord(42, "My Random Word Test",
  "Tests for Random Words", self.test_settings, app)

self.tests = dict([
  (tICMP.id, tICMP), ..., (tRandomWord.id, tRandomWord)])
```