Testing IPv6 Firewalls with ft6

Oliver Eggert

IPv6 Security Summit @ TROOPERS14

March 17th, 2014
Outline

1. The beginnings
2. Design of ft6
3. Tests done by ft6
4. Live Demo
5. Testing ip6tables
6. Pitfalls
7. (optionally: writing your own tests)
The beginnings

- “IPv6 Intrusion Detection System” Project (2011 - 2013)
- at University of Potsdam
- funded by “Bundesministerium für Bildung und Forschung”
IPv6 adoption continues to rise. However, lack of IPv6-enabled tools for:

- analyzing threat level
- checking firewall/IDS configuration
- checking firewall/IDS capabilities
- checking IPv6 “readiness”
The beginnings: contributions

**IPv6 Darknet**

- not advertised
- check who's scanning
- /48 network
- less than 1200 packets in 9 months
The beginnings: contributions

IPv6 Honeypot

- check what attackers are doing
- honeydv6
- emulates a whole virtual network and services
IPv6 IDS

- Snort-Plugin: check Martin Schütte’s talk tomorrow!
The beginnings: contributions

Performance Benchmarks

- how well does IPv6-enabled hardware perform?
- what impact do “additional” features of IPv6 have?
The beginnings: information

- project is over now
- check results and publications at:

www.idsv6.de
Outline

1. The beginnings
2. Design of ft6
3. Tests done by ft6
4. Live Demo
5. Testing ip6tables
6. Pitfalls
7. (optionally: writing your own tests)
Design of ft6 – motivation

Why should I?

- you are responsible for the network
- you have to / want to migrate to IPv6
- can you switch now?
- not really a good methodology for
  - telling if your device supports IPv6
  - comparing firewalls
  - finding problems in your configuration
Design of ft6 – motivation

checking for “Supports IPv6”-stickers is not an option!
Design of ft6 – motivation

- you **must** check for yourself
- why is this hard?
- lot of SHOULDs, MUSTs and REQUIREDs for IPv6
- across lot of different RFCs
- vague
- best practices
- still evolving, hard to keep track
Design of ft6 – motivation

- ft6 should help
- goal: easy to configure and visualize results
- easily reproducible, comparable
- usually tedious, error prone work
- provide starting point for firewall evaluation
- also consider testing for performance, attacks on local network
- can act as a framework for new tests
Design of ft6 – Architecture

- ft6 is a client-server application

Firewall
Design of ft6 – Architecture

- ft6 is a client-server application
- requires machines on both sides of your firewall
Design of ft6 – Architecture

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

- ft6 is a client-server application
- requires machines on both sides of your firewall
- place machines not more than one hop away from firewall
- one open port
Client and Server perform a handshake
- Server begins to sniff
- Some packets pass the firewall
- Others are dropped
Design of ft6 – Running ft6

- Client and Server perform a handshake
- Server begins to sniffs
- Client starts sending packets
Design of ft6 – Running ft6

- Client and Server perform a handshake
- Server begins to sniffs
- Client starts sending packets
- Some packets pass the firewall
Design of ft6 – Running ft6

- Client and Server perform a handshake
- Server begins to sniffs
- Client starts sending packets
- Some packets pass the firewall
- Others are dropped
Design of ft6 – Running ft6

- Server sends back list of packets it received

Client → Firewall: “I saw packet #1“ → Server

Oliver Eggert
Testing IPv6 Firewalls with ft6
Design of ft6 – Running ft6

- Server sends back list of packets it received
- Client figures out what went missing and displays result
Design of ft6 – Design of ft6

- open-source (Creative Commons BY-NC-SA 3.0)
- uses scapy 2.2.0, python 2.7, PyQt 4
- developed on (2.6.32), tested with more recent (3.7.1)
- should work on Windows 7, Mac OS X
1. The beginnings
2. Design of ft6
3. Tests done by ft6
4. Live Demo
5. Testing ip6tables
6. Pitfalls
7. (optionally: writing your own tests)
Tests done by ft6

- written by EANTC
- ft6 does 9 of those
Tests done by ft6 – Test 1: ICMPv6 filtering

- Check if the firewall correctly forwards and discards ICMPv6 Packets.
- Informational, diagnostic messages
- Identified by type and code field.

```
+-----------------+-----------------+-------------------+
| Type     | Code | Checksum |
+-----------------+-----------------+-------------------+
|                |                |                    |
| + Message Body + |
+-----------------+-----------------+-------------------+
```

- RFC 4890 “Recommendations for Filtering ICMPv6 Messages in Firewalls”
Tests done by ft6 – Test 1: ICMPv6 filtering

belong to one of 3 groups: mandatory, optional and nonfiltered

1 mandatory: Always reject them.
local Messages (Neighbor Solicitation, Neighbor Advertisement).

2 optional: Reject unless needed.
unassigned types and codes
3 (Code 1), 4 (Code 0), 4–99, 102 – 126, 144 – 147, 150, 154

3 nonfiltered: Always forward them.
ecessary for correct operation (echo request, echo reply, packet too big).
1, 2, 3 (Code 0), 4 (Code 1), 4 (Code 2), 128, 129.
Tests done by ft6 – Test 2: Routing Header

- Check if the firewall correctly forwards and discards packets containing a Routing Header.
- Used to specify a list of nodes a packet has to visit
- RFC 5095 “Deprecation of Type 0 Routing Headers in IPv6”
- loops $\rightarrow$ denial of service
- applies to Routing Header type 0 only
Tests done by ft6 – Test 2: Routing Header

- Depends upon type and segments-left field.

<table>
<thead>
<tr>
<th>Next Header</th>
<th>Hdr Ext Len</th>
<th>Routing Type</th>
<th>Segments Left</th>
</tr>
</thead>
<tbody>
<tr>
<td>type-specific data</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

+------------------------------------------------------------------+
| | | | |
+------------------------------------------------------------------+
Tests done by ft6 – Test 2: Routing Header

<table>
<thead>
<tr>
<th>RH-type</th>
<th>segs. left</th>
<th>action</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>$\neq 0$</td>
<td>drop</td>
</tr>
<tr>
<td>2</td>
<td>$\neq 1$</td>
<td>drop</td>
</tr>
<tr>
<td>others</td>
<td>$\neq 0$</td>
<td>drop</td>
</tr>
</tbody>
</table>
Tests done by ft6 – Test 3: Chained Extension Headers

- Check if the firewall correctly forwards and discards packets containing a number of different Extension Headers.
- DSTOPT header at most twice (before a RH, before Layer 4)
- HBH Options only after base IPv6 header
- others: at most once (should)
- avoid ambiguity, prevent denial of service
Tests done by ft6 – Test 3: Chained Extension Headers

1. A single DSTOPT. Should be forwarded.
2. Two DSTOPTs in a row. Should be dropped.
3. DSTOPT, Routing Header, DSTOPT. Forward.
4. A single HBH. Forward.
5. Two HBHs in a row. Drop.
6. A DSTOPT, followed by one HBH. Drop.
7. HBH, DSTOPT, Routing Header, HBH. Drop.
Tests done by ft6 – Test 4: Overlapping Fragments

- Check if the firewall correctly detects overlapping fragments
- Forward only if no overlap
- RFC 5722 “Handling of Overlapping IPv6 Fragments”
- IPv6 fragments at source & reassembles at destination
- fragments controlled by fragment-offset
Tests done by ft6 – fragmentation

- source want's to send a packet that is too big

![Diagram of packet structure with header and payload]
Tests done by ft6 – fragmentation

- payload is split into chunks
- each sent as separate *fragment*

```
+-------------------+
|       Header      |
+-------------------+
| Fragment #1       |
| Fragment #2       |
| Fragment #3       |
```
Tests done by ft6 – reassembly

- destination receives fragment one, allocates buffer

Fragment #1
Tests done by ft6 – reassembly

- virtually splits buffer into slots

<table>
<thead>
<tr>
<th>Slot #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slot #2</td>
</tr>
<tr>
<td>Slot #3</td>
</tr>
</tbody>
</table>
“target-slot” is determined by fragment-offset

| Slot #1  |  
|----------|----------
| Slot #2  |  
| Slot #3  |  

Fragment #1

offset: 1
Tests done by ft6 – reassembly

- data is copied accordingly

<table>
<thead>
<tr>
<th>Fragment #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slot #2</td>
</tr>
<tr>
<td>Slot #3</td>
</tr>
</tbody>
</table>
Tests done by ft6 – reassembly

- same procedure for second fragment

Fragment #3

<table>
<thead>
<tr>
<th>Fragment #1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slot #2</td>
</tr>
<tr>
<td>Slot #3</td>
</tr>
</tbody>
</table>
Tests done by ft6 – reassembly

- same procedure for second fragment
Tests done by ft6 – reassembly

- same procedure for third fragment

- Fragment #2
- Fragment #1
- Slot #2
- Fragment #3
Tests done by ft6 – reassembly

- same procedure for third fragment

- Fragment #1
- Fragment #2
- Fragment #3
Tests done by ft6 – reassembly

- all fragments have arrived

Fragment #1

Fragment #2

Fragment #3
Tests done by ft6 – reassembly

- reassembly complete

Payload
Tests done by ft6 – attack

- first fragment arrives. It carries a TCP-Header

<table>
<thead>
<tr>
<th>Slot #1</th>
<th>TCP - port: 80</th>
<th>offset: 1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slot #2</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Slot #3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Tests done by ft6 – attack

- data is copied into buffer

<table>
<thead>
<tr>
<th>TCP - port: 80</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slot #2</td>
</tr>
<tr>
<td>Slot #3</td>
</tr>
</tbody>
</table>
Tests done by ft6 – attack

- second fragment has “wrong” offset

<table>
<thead>
<tr>
<th>TCP - port: 22</th>
<th>offset: 1</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>TCP - port: 80</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slot #2</td>
</tr>
<tr>
<td>Slot #3</td>
</tr>
</tbody>
</table>
Tests done by ft6 – attack

- still, data get’s copied

<table>
<thead>
<tr>
<th>TCP - port: 22</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slot #2</td>
</tr>
<tr>
<td>Slot #3</td>
</tr>
</tbody>
</table>
Tests done by ft6 – attack

- last fragment arrives

<table>
<thead>
<tr>
<th>Fragment #3</th>
<th>offset: 2</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>TCP - port: 22</th>
</tr>
</thead>
<tbody>
<tr>
<td>Slot #2</td>
</tr>
<tr>
<td>Slot #3</td>
</tr>
</tbody>
</table>
Tests done by ft6 – attack

- data get’s copied

```
<table>
<thead>
<tr>
<th>TCP - port: 22</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fragment #3</td>
</tr>
<tr>
<td>Slot #3</td>
</tr>
</tbody>
</table>
```
Tests done by ft6 – attack

- reassembly complete, firewall bypassed

TCP - port: 22

Fragment #3
Tests done by ft6 – Test 4: Overlapping Fragments

1. Fragments w/o overlap. Should be forwarded.
2. Overlapping fragments that overwrite the TCP-port. Drop
3. Overlapping fragments that overwrite the payload. Drop
Tests done by ft6 – Tests 5 & 6: Tiny IPv6 Fragments

- Check if the firewall inspects the second fragment if no Layer 4 is present in the first fragment
- Firewall should wait for next fragment before deciding
- Check if the firewall respects the timeout as specified in the rfc
- prevent resource starvation
- allow for “some” lag
- drop after 60 seconds
Tests done by ft6 – Tests 5 & 6: Tiny IPv6 Fragments

1. Tiny-Fragment with allowed port in second fragment. Forward.
2. Tiny-Fragment with denied port in second fragment. Drop.
3. Send first fragment, wait 59 seconds, send last fragment. Forward.
4. Send first fragment, wait 61 seconds, send last fragment. Drop.
Tests done by ft6 – Test 7: Excessive HBH/DSTOPT Options

- Check if the firewall blocks packets with multiple options
- IPv6 supports different option types per header
  | Option Type | Opt Data Len | Option Data |
  +----------------+---------------+-------------|
  +----------------+---------------+-------------|
- can be daisy-chained
- Most option types should occur at most once
- Only Pad1 and PadN are allowed multiple times
- prevent ambiguity, prevent denial of service
- RFC 4942 “IPv6 Transition/Coexistence Security Considerations”
Tests done by ft6 – Test 7: Excessive HBH/DSTOPT Options

Each variant has duplicate options. Each should be dropped.

1. Jumbo Payload, PadN, Jumbo Payload.
2. Router Alert, Pad1, Router Alert
3. Quick Start, Tunnel Encapsulation Limit, PadN, Quick Start
4. RPL Option, PadN, RPL Option
Tests done by ft6 – Test 8: PadN Covert Channel

- Check if the firewall can block packets with non-zero padding
- Used to align options
- (usually) don’t carry a payload
- RFC 4942 “IPv6 Transition/Coexistence Security Considerations”
Tests done by ft6 – Test 8: PadN Covert Channel

1. Padding with payload all zeroes in a HBH. Forward.
2. Padding with other payload in a HBH. Drop.
3. Padding with payload all zeroes in a DSTOPT. Forward.
4. Padding with other payload in a DSTOPT. Drop.
Tests done by ft6 – Test 9: Address Scopes

- Verify that the firewall does not route traffic from an inappropriate scope.
  1. ff00::/16 (multicast)
  2. fe80::/10 (link local)
- RFC 4942 “IPv6 Transition/Coexistence Security Considerations”
- 256 packets with source addresses from group 1. Drop.
- 16 packets with source addresses from group 2. Drop.
Outline

1. The beginnings
2. Design of ft6
3. Tests done by ft6
4. Live Demo
5. Testing ip6tables
6. Pitfalls
7. (optionally: writing your own tests)
Live Demo

Live Demo
Outline

1. The beginnings
2. Design of ft6
3. Tests done by ft6
4. Live Demo
5. Testing ip6tables
6. Pitfalls
7. (optionally: writing your own tests)
Testing ip6tables – setup

- Linux grml 3.7.1-grml-amd64 Debian 3.7.9+grml.1 x86_64
- ip6tables 1.4.18
- ft6 2013-07-28
- Python 2.7.3
- Scapy 2.2.0
Testing ip6tables – procedure

- use a “default” configuration
- perform test
- if test fails:
  - try to improve config
  - perform test again
Testing ip6tables – default configuration

```
ip6tables -A FORWARD -p tcp --dport 80 -j ACCEPT
ip6tables -A FORWARD -p udp --dport 80 -j ACCEPT
ip6tables -A FORWARD -m state --state ESTABLISHED,RELATED -j ACCEPT
ip6tables -P FORWARD DROP
```
## Testing ip6tables – Results

<table>
<thead>
<tr>
<th>test</th>
<th>basic config</th>
<th>improved rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMPv6 Filtering</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Routing Header</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>Header Chain</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Overlapping Fragments</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Tiny IPv6 Fragments</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Excessive HBH Options</td>
<td>✗</td>
<td>✓</td>
</tr>
<tr>
<td>PadN Covert Channel</td>
<td>✗</td>
<td>✗</td>
</tr>
<tr>
<td>Address Scope</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>
Check how the firewall handles ICMPv6-messages according to the three groups mandatory, optional, nonfiltered.

default config

✓ mandatory: all dropped
✓ optional: all dropped
✗ nonfiltered: all dropped
Testing ip6tables – Test 1: ICMPv6 Filtering

Cause:

policy is DROP

improved configuration:

ip6tables -A FORWARD -p icmpv6 --icmpv6-type destination-unreachable \ -j ACCEPT

ip6tables -A FORWARD -p icmpv6 --icmpv6-type packet-too-big \ -j ACCEPT

ip6tables -A FORWARD -p icmpv6 --icmpv6-type ttl-zero-during-transit \ -j ACCEPT
Testing ip6tables – Test 1: ICMPv6 Filtering

improved configuration (cont.):

```bash
ip6tables -A FORWARD -p icmpv6 --icmpv6-type unknown-header-type \ -j ACCEPT

ip6tables -A FORWARD -p icmpv6 --icmpv6-type unknown-option \ -j ACCEPT

ip6tables -A FORWARD -p icmpv6 --icmpv6-type echo-request \ -m limit --limit 900/min -j ACCEPT

ip6tables -A FORWARD -p icmpv6 --icmpv6-type echo-reply \ -m limit --limit 900/min -j ACCEPT
```
Testing ip6tables – Test 1: ICMPv6 Filtering

improved config

✓ mandatory: all dropped
✓ optional: all dropped
✓ nonfiltered: all forwarded
Testing ip6tables – Test 1: ICMPv6 Filtering

other firewalls

- most allow filtering by type and code
- some drop packets even if they are allowed (see pitfalls).
Testing ip6tables – Test 2: Routing Header

Check how the firewall handles Routing Headers. Depends on \textit{RH type} and \textit{segments left}.

\begin{itemize}
  \item \texttt{default config}
  \begin{itemize}
    \item \checkmark \texttt{type = 0, segments-left = 0}: forwarded
    \item \xmark \texttt{type = 0, segments-left \neq 0}: forwarded
    \item \xmark \texttt{type = 2, segments-left \neq 1}: forwarded
    \item \checkmark \texttt{type = 2, segments-left = 1}: forwarded
    \item \checkmark \texttt{type = 200, segments-left = 0}: forwarded
    \item \xmark \texttt{type = 200, segments-left \neq 0}: forwarded
  \end{itemize}
\end{itemize}
Testing ip6tables – Test 2: Routing Header

Cause:
- Packets are directed at allowed port 80
- ip6tables does not check the Routing Header

Solution:
Use the ip6tables-module rt. But **not** like this:

```
ip6tables -A FORWARD -m rt --rt-type 0 -j ACCEPT
```

This will accept all packets containing a routing header w/o checking for the port. Better: handle RH in separate chain.
Testing ip6tables – Test 2: Routing Header

improved configuration:

```bash
ip6tables -N routinghdr
ip6tables -A routinghdr -m rt --rt-type 0 ! --rt-segsleft 0 -j DROP
ip6tables -A routinghdr -m rt --rt-type 2 ! --rt-segsleft 1 -j DROP
ip6tables -A routinghdr -m rt --rt-type 0 --rt-segsleft 0 -j RETURN
ip6tables -A routinghdr -m rt --rt-type 2 --rt-segsleft 1 -j RETURN
ip6tables -A routinghdr -m rt ! --rt-segsleft 0 --j DROP

ip6tables -A FORWARD -m ipv6header --header ipv6-route --soft \
   -j routinghdr
```
Testing ip6tables – Test 2: Routing Header

**improved configuration**

- ✓ type = 0, segments-left = 0: forwarded
- ✓ type = 0, segments-left ≠ 0: dropped
- ✓ type = 2, segments-left ≠ 1: dropped
- ✓ type = 2, segments-left = 1: forwarded
- ✓ type = 200, segments-left = 0: forwarded
- ✓ type = 200, segments-left ≠ 0: dropped
Testing ip6tables – Test 2: Routing Header

other firewalls

- some can only drop *all* or *no* RH
- some can only inspect *type*, not *segments-left*
Testing ip6tables – Test 3: Extension Header Chain

Check how the firewall handles packets containing header chains.

default config

✔ DSTOPT: forwarded
✗ HBH: dropped
✗ DSTOPT-HBH: forwarded
✗ DSTOPT-DSTOPT: forwarded
✔ HBH-HBH: dropped
✔ DSTOPT-RH-DSTOPT: forwarded
✔ HBH-DSTOPT-RH-HBH: dropped
Testing ip6tables – Test 3: Extension Header Chain

Cause:
- Packets are directed at allowed port 80
- ip6tables does not check the DSTOPT or HBH

Solution:
Use ip6tables-module ipv6header or ipv6headerorder. Check headers in separate chain. Problems:
- Not stateful enough – need to enumerate all possible combinations
- Rule for forwarding single HBH didn’t work.
- Dropping duplicate DSTOPT-DSTOPT also drops single DSTOPT.
other firewalls

- similar problems
- only a Cisco ASA w/ additional IPS-module could detect duplicates.
- performance benchmark showed importance:
- throughput down to approx. 65 when sending extension headers
Testing ip6tables – Test 4: Overlapping Fragments

Check how the firewall handles packets containing fragments.

**default config**

- ✓ no overlap: forwarded
- ✓ overlap overwriting the port: dropped
- ✓ overlap overwriting the payload: dropped
Testing ip6tables – Test 4: Overlapping Fragments

other firewalls

- some do not allow fragments at all
- some do not recognize overlap
Testing `ip6tables` – Tests 5 & 6: Tiny Fragments

Check how the firewall handles “Tiny Fragments” (upper layer header not present in first fragment). Check if timeout of 60s is handled correctly.

---

**default config**

- ✗ port 80 in second fragment (allowed): dropped
- ✓ port 22 in second fragment (forbidden): dropped
- ✗ last fragment arriving after 59 seconds: dropped
- ✓ last fragment arriving after 61 seconds: dropped
Testing ip6tables – Tests 5 & 6: Tiny Fragments

Problem:

- ip6tables appears to drop all tiny fragments
- checking for timeout is not useful.
- no solution was found, no improved config.

other firewalls

- similar results
Testing ip6tables – Test 7: Excessive HBH/DSTOPT Options

Check how the firewall handles extension headers containing duplicate options.

default config

✓ HBH with Jumbo-PadN-Jumbo: dropped
✗ DSTOPT with Jumbo-PadN-Jumbo: forwarded
✗ HBH with RouterAlert-Pad1-RouterAlert: forwarded
✗ DSTOPT with RouterAlert-Pad1-RouterAlert: forwarded
✗ HBH with QuickStart-TunnelEncapLimit-PadN-QuickStart: forwarded
✗ DSTOPT with QuickStart-TunnelEncapLimit-PadN-QuickStart: forwarded
✓ HBH with RPL-PadN-RPL: dropped
✗ DSTOPT with RPL-PadN-RPL: forwarded
Testing ip6tables – Test 7: Excessive HBH/DSTOPT Options

Cause:
not really sure...

Solution:

- Use ip6tables-modules hbh and dst to check for payload of these headers.

- enumerate *all possible combinations*

- example config for dropping second combination:

  ```
ip6tables -A FORWARD -m dst --dst-opts 194,1,194 -j DROP
  ```
Testing ip6tables – Test 7: Excessive HBH/DSTOPT Options

improved config

- ✔ HBH with Jumbo-PadN-Jumbo: dropped
- ✔ DSTOPT with Jumbo-PadN-Jumbo: dropped
- ✔ HBH with RouterAlert-Pad1-RouterAlert: dropped
- ✔ DSTOPT with RouterAlert-Pad1-RouterAlert: dropped
- ✔ HBH with QuickStart-TunnelEncapLimit-PadN-QuickStart: dropped
- ✔ DSTOPT with QuickStart-TunnelEncapLimit-PadN-QuickStart: dropped
- ✔ HBH with RPL-PadN-RPL: dropped
- ✔ DSTOPT with RPL-PadN-RPL: dropped
Testing ip6tables – Test 7: Excessive HBH/DSTOPT Options

other firewalls

- most were unable to inspect contents of options
- some were only able to inspect the first option
Check how the firewall handles packets containing a PadN-option. It’s payload should be all zeroes.

**default config**

- ✗ HBH with PadN-payload = 0: dropped
- ✓ HBH with PadN-payload ≠ 0: dropped
- ✓ DSTOPT with PadN-payload = 0: forwarded
- ✗ DSTOPT with PadN-payload ≠ 0: forwarded
Testing ip6tables – Test 8: PadN Covert Channel

Cause:
- ip6tables doesn’t seem to check payload at all
- no solution found, no improved config.

Other firewalls
- same result
Testing ip6tables – Test 9: Address Scopes

Check how the firewall handles packets originating from an inappropriate scope.

**default config**

- multicast: all dropped
- link-local: all dropped

**other firewalls**

- same result
# Testing ip6tables – conclusion

<table>
<thead>
<tr>
<th>test</th>
<th>improved rules</th>
</tr>
</thead>
<tbody>
<tr>
<td>ICMPv6 Filtering</td>
<td>✓</td>
</tr>
<tr>
<td>Routing Header</td>
<td>✓</td>
</tr>
<tr>
<td>Header Chain</td>
<td>✗</td>
</tr>
<tr>
<td>Overlapping Fragments</td>
<td>✓</td>
</tr>
<tr>
<td>Tiny IPv6 Fragments</td>
<td>✗</td>
</tr>
<tr>
<td>Excessive HBH Options</td>
<td>✓(^1)</td>
</tr>
<tr>
<td>PadN Covert Channel</td>
<td>✗</td>
</tr>
<tr>
<td>Address Scope</td>
<td>✓</td>
</tr>
</tbody>
</table>

\(^1\) not very elegant
Outline

1. The beginnings
2. Design of ft6
3. Tests done by ft6
4. Live Demo
5. Testing ip6tables
6. Pitfalls
7. (optionally: writing your own tests)
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
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
Pitfalls

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

- don’t focus too hard on *rfc-conformity*
- 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
Pitfalls

results:

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

1. The beginnings
2. Design of ft6
3. Tests done by ft6
4. Live Demo
5. Testing ip6tables
6. Pitfalls
7. (optionally: writing your own tests)
Writing your own test

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

1. create a class for your test
2. craft packets in the prepare method
3. register your test with the application

(More detailed in ft6’s documentation)
Writing your own tests – packet handling with scapy

- handling network packets is usually messy
  - binary protocols
  - accessing individual flags involves bitshifting or bitmasking
- sending and receiving is error-prone, too
- scapy does all that for you and is human readable.
- great TAB-completion
Writing your own tests – packet handling with scapy
Writing your own tests – packet handling with scapy

```python
>>> mypacket = IPv6()
>>> mypacket.show()
### IPv6 ###
  version= 6
tc= 0
fl= 0
plen= None
nh= No Next Header
hlim= 64
src= ::1
dst= ::1
```
Writing your own tests – packet handling with scapy

```
>>> mypacket.src = "2001:db8::abcd"
>>> mypacket.show()
### [ IPv6 ]###
   version= 6
tc= 0
fl= 0
plen= None
nh= No Next Header
hlim= 64
src= 2001:db8::abcd
dst= ::1
```
Writing your own tests – packet handling with scapy

>>> tcp=TCP(dport=80)
>>> payload = "GET index.html HTTP/1.1"
>>> result = mypacket/tcp/payload
Writing your own tests – packet handling with scapy

```python
>>> result.show2()
### [ IPv6 ]###
  version= 6L
tc= OL
ttl= 64
fl= OL
plen= 43
nh= TCP
hlim= 64
src= 2001:db8:abcd
dst= ::1
### [ TCP ]###
sport= ftp_data
dport= www
seq= 0
ack= 0
dataofs= 5L
reserved= 0L
flags= S
window= 8192
chksum= 0xd79d
urgptr= 0
options= []
### [ Raw ]###
  load= 'GET index.html HTTP/1.1'
```
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)

(copy-paste, change the name)
Writing your own tests

Step 2: Craft packets in the prepare-method

def prepare(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)

    p = Ft6Packet(e/ip/udp/Raw("randomword"))
    p.setValid()
    p.setDescription("A valid packet containing a random word")
    p.ifDropped("This violates rfc #23")
    self.addPacket(p)
Writing your own tests

Step 2: Craft packets in the prepare-method

```python
p = Ft6Packet(e/ip/udp/Raw("otherword"))
p.setInvalid()
p.setDescription("An invalid packet containing some other word")
p.ifForwarded("This violates rfc #42")
self.addPacket(p)
```
Writing your own tests

- That’s it!
- ft6 will send each packet that has been added like this
- and add results according to the packet’s state
Writing your own tests

Step 3: register your test

```python
# create test classes, store them in the dictionary
# so they can later be called by their id
ICMP = TestICMP(1, "ICMPv6 Filtering", "The ICMP Test",
                 self.test_settings, app)
self.registerTest(ICMP)

...

RandomWord = TestRandomWord(42, "My Random Word Test",
                            "Tests for Random Words", self.test_settings, app)
self.registerTest(RandomWord)
```
Wrap up

- a lot of things to take care of
- don’t trust the vendors
- also do performance, link-local tests
- ft6 is a work in progress
- lots of improvement could be done
- better results
- more tests
Thank You! Questions?

- get ft6 from: https://github.com/olivereggert/ft6
- more info on the project: www.idsv6.de