testing your IPv6-firewall with ft6

Oliver Eggert

Potsdam University
Institute for Computer Science
Operating Systems and Distributed Systems

Potsdam, July 30, 2013

Agenda

1. idsv6, origin of ft6
2. design of ft6
3. test specification
4. live demo
5. *(optionally: writing your own tests)*

idsv6

- "IPv6 Intrusion Detection System" Project (2011 - 2013)
- funded by "Bundesministerium für Bildung und Forschung"

Universität Potsdam

BEUTH HOCHSCHULE
FÜR TECHNIK
BERLIN
University of Applied Sciences

EANTC

STRATO

- lack of IPv6-enabled tools for
  - analyzing threat level
  - checking firewall/IDS configuration
  - checking firewall/IDS capabilities
  - checking IPv6 "readiness"

idsv6: contributions

- IPv6 Darknet

  $/48$-net $<$ 1200 packets in 9 months
- IPv6 Honeypot

  honeydv6
- IPv6 Plugin for Snort

  maintains network state, allows signatures for IPv6 header fields
- load tests

  done by eantc
- protocol tests

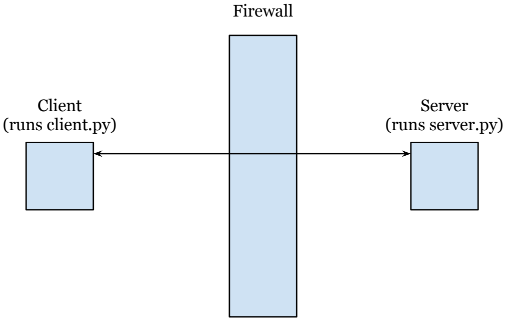
  ft6

`www.idsv6.de`

motivation

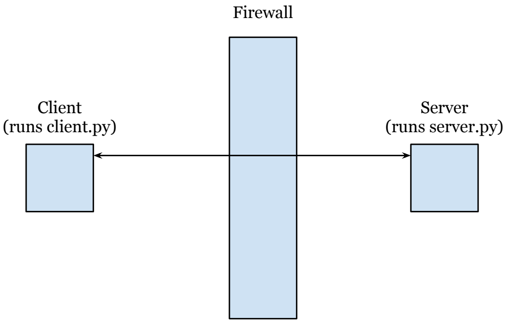
- say you are...
  - new to IPv6
  - try to improve your firewall config
  - try to compare firewalls
  - ...
- lot of `SHOULD`s, `MUST`s and `REQUIRED`s for IPv6
- across lot of different `RFC`s
- vague
- best practices
- hard to keep track
- EANTC wrote a sepcification
- ft6 implements 9 of those tests

ft6 – Architecture



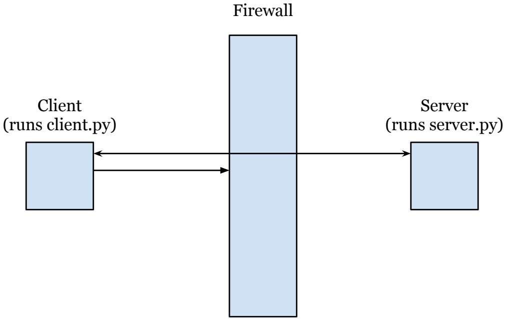
- ft6 is a client-server application
- requires machines on both sides of your firewall
- one open port
- place machines not more than one hop away from firewall

Running ft6



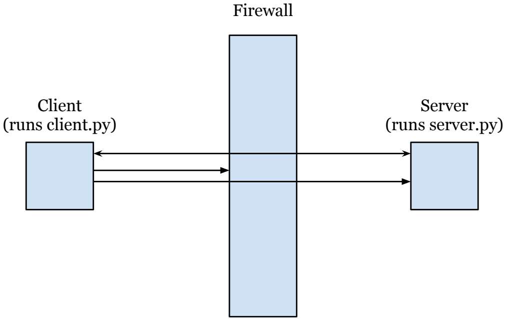
- Client and Server exhange control messages
  - Start / End / Results

Running ft6



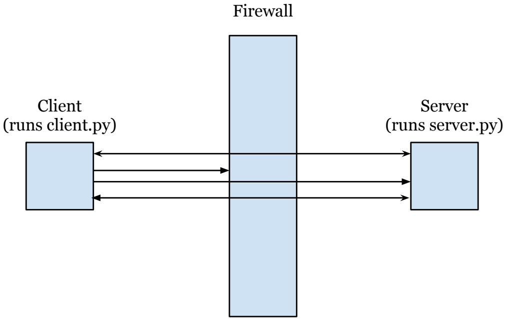
- Client sends packets
- Server sniffs

Running ft6



- Client sends packets
- Server sniffs

Running ft6



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

Design of ft6

- goal: easy to configure and visualize results
- open-source (Creative Commons BY-NC-SA 3.0)
- uses scapy 2.2.0, python 2.7, PyQt 4
- developed on Debian Linux 6 (2.6.32), tested with more recent grml (3.7.1)
- *should* work on Windows 7, Mac OS X
- can act as a framework for new tests

Test cases

test 1: ICMPv6 filtering

- Check if the firewall correctly forwards and discards ICMPv6 Packets.
- Depends upon `type` and `code` field.
- Categories *mandatory*, *optional* and *nonfiltered*
- RFC 4890 "Recommendations for Filtering ICMPv6 Messages in Firewalls"

test 2: Routing Header

- Check if the firewall correctly forwards and discards packets containing a Routing Header.
- Depends upon `type` and `segments-left` field.
- RFC 5095 "Deprecation of Type 0 Routing Headers in IPv6"

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)
- RFC 2460 "Internet Protocol, Version 6 (IPv6) Specification"

test 4: Overlapping Fragments

- Check if the firewall correctly detects overlapping fragments
- Forward only if no overlap
- RFC 5722 "Handling of Overlapping IPv6 Fragments"

tests 5 and 6: Tiny IPv6 Fragments Timeout

- Check if the firewall can inspect the second fragment if no Layer 4 is present within the first fragment
- http://tools.ietf.org/id/draft-gont-6man-oversized-header-chain-02.txt
- Check if the firewall respects the timeout as specified in the rfc
- drop after 60 seconds
- RFC 2460 " Internet Protocol, Version 6 (IPv6) Specification"

test 7: Excessisve HBH/DSTOPT Options

- Check if the firewall blocks packets with multiple options
- Most options should occur at most once
- Only Pad1 and PadN are allowed multiple times
- RFC 4942 "IPv6 Transition/Coexistence Security Considerations"

test 8: PadN Covert Channel

- Check if the firewall can block packets with non-zero padding
- RFC 4942 "IPv6 Transition/Coexistence Security Considerations"

test 9: Adress Scopes

- Verify that the firewall does not route traffic from an inappropriate scope.
- `ff00::/16` and `fe80::/10`
- RFC 4942 "IPv6 Transition/Coexistence Security Considerations"

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":
  ⇒ no longer red
- can’t make it green:
  ⇒ 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

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`

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)

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: 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)
```

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
```

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
```

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)
self.registerTest(tICMP)

...

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