The IPv6 Snort Plugin

Martin Schütte

18 March 2014
Context

- Diploma thesis
- 2011 at Potsdam University
- part of “attack prevention and validated protection of IPv6 networks”
State ~ 1994

IPv4 Internet:

- Research and Academic Networks
- Known design & implementation errors
- Little experience with protocol security
- No urgency for improvement
State ~ today

IPv6 Internet:

- Research and Academic Networks
- Known design & implementation errors
- Little experience with protocol security
- No urgency for improvement (?)
IPv6 Security Issues

- Main IPv6 RFCs from 1995/1998
  ⇒ many years of IPv4 security experience to catch up with

  - Many accompanying RFCs and Internet Drafts (IPsec, SEND, RH0 deprecation, RA Guard, …)
  - Few (yet already old) implementations
  - Very little in end user devices
  - Uncertainty hinders deployment
Attacks Against IPv6

The usual:
- Value ranges
- Fragmentation
- Denial of Service
- Portscans
- Errors in Application Layer

IPv6 specific:
- Variable headers
- Multicast
- Routing
- v4/v6 Transition
- Autoconfiguration
- Neighbor Discovery
Header Chaining

IPv6

Next Header

ICMPv6

IPv6

Next Header

TCP

Data

IPv6

Next Header

ESP

UDP

Data

IPv6

Routing Header

Fragment Header

ICMPv6

IPv6 Snort Plugin

2014-03-18 7 / 41
Design Flaw

Designed in 1994, same premise as IPv4: secure and trustworthy LAN
⇒ cable LAN in organizational hierarchy

No consideration of:

- WiFi
- mobile usage
- anonymous users
Local Attacks

Simple Denial of Service:

1. Host Alice starts *Duplicate Address Detection*: “Anyone using IP X?”
2. Host Eve answers “I have IP X.”
3. goto 1

Routing/Man in the Middle:

1. Host Eve sends ICMPv6 Redirect: “This is router Bob, for google.com please use router Eve.”
Remote Attacks

• Denial of Service
  • Neighbor Cache Exhaustion
  • Oversized IPv6 Header Chains
  • Excessive Hop-by-Hop Options

• Routing
  • RH0 source routing
  • Loop using IPv6 Automatic Tunnels
Attack Collections: THC Toolkit and SI6 Networks’ IPv6 Toolkit

Tools/Attacks/Tests for:

- Autoconfiguration DoS
- Neighbor Cache
- Routing/Redirect
- Flood-Attacks
- Multicast Listener Discovery
- DHCPv6
- implementation6
Countermeasures

• Filter known-bad packets
• Show anomalous network activity
• Collect data for correlation and detection
Where to Monitor

Placement at:
- Routers
- Switches
- Packet Filters
- Hosts

Implementation as:
- Stand-alone tool
- Add-on for existing application
- Operating System module

⇒ High versatility: Intrusion Detection Systems
Target System: Snort 2.9

- Widely used Open Source NIDS
- Filter/inline mode (Intrusion Prevention System)
- Plugin APIs
- Decoder for common tunnel protocols
IPv6
IDS/Snort
IPv6 Plugin
Conclusion

Snort Packet Processing Overview

Network → DAQ/libpcap → Packet Decoder → Pre-processor → Detection Engine → Alert, Log Output → Logfiles, Database

Rules

Snort

Martin Schütte
IPv6 Snort Plugin
2014-03-18
Decoding

Incoming Packet

- DecodeEthPkt
  - Ethernet
    - DecodeVlanPkt
      - 802.1Q
    - DecodePPPoEPkt
      - PPPoE
    - DecodePPpPktEncapsulated
      - PPP
    - DecodeIP
      - IPv4
        - DecodeICMP
          - ICMP
        - DecodeUDP
          - UDP
    - DecodeIPv6
      - IPv6
        - DecodeIPv6Extensions
          - IPv6 Ext Hdrs
        - DecodeIPv6Options
          - IPv6 Options
    - DecodeARP
      - ARP
    - DecodeICMP6
      - ICMPv6

Martin Schütte
IPv6 Snort Plugin
2014-03-18 16 / 41
Decoding Result: `struct _Packet`

```c
typedef struct _Packet {
    const DAQ_PktHdr_t *pkth; // packet meta data
    const uint8_t *pkt; // raw packet data

    EtherARP *ah;
    const EtherHdr *eh; /* standard TCP/IP/Ethernet/ARP headers */
    const VlanTagHdr *vh;

    const IPHdr *iph, *orig_iph; /* and orig. headers for ICMP_*_UNREACH */
    const IPHdr *inner_iph; /* if IP-in-IP, this will be the inner */
    const IPHdr *outer_iph; /* if IP-in-IP, this will be the outer */

    uint32_t preprocessor_bits; /* flags for preprocessors to check */
    uint32_t preproc_reassembly_pkt_bits;

    uint8_t ip_option_count; /* number of options in this packet */
    uint8_t tcp_option_count;
    uint8_t ip6_extension_count;
    uint8_t ip6_frag_index;

    IPOptions ip_options[MAX_IP_OPTIONS];
    TCPOptions tcp_options[MAX_TCP_OPTIONS];
    IP6Extension ip6_extensions[MAX_IP6_EXTENSIONS];

    // ...
} Packet;
```
Rule Engine

Example detection rule:

```plaintext
var EXTERNAL_NET any
var SMTP_SERVERS [192.0.2.123, 2001:db8:12:ab::123]

alert tcp $EXTERNAL_NET any -> $SMTP_SERVERS 25 (
    flow:to_server,established;
    content: "|0A|Croot|0A|Mprog";
    metadata:service smtp;
    msg:"SMTP sendmail 8.6.9 exploit";
    reference:bugtraq,2311;reference:cve,1999-0204;
    classtype:attempted-user;
    sid:669; rev:9;
)
```
IPv6 Support

technically yes, but …

All major IDS have IPv6 support.

What does that mean?

• Fragment reassembly
• TCP & UDP decoding ⇒ upper-layer checks
• Decoder-warning on severe protocol errors

Not:

• check extensions (Routing Headers, Jumbograms)
• support all rule options (fragbits)
• IPv6 specific detection (ICMPv6/Neighbor Discovery)
IPv6 Signatures

Existing rules work for IPv4 and IPv6

No keywords for IPv6-only fields, no IPv6-only rules provided

```
alert ip icmp any -> any any 
  (msg:"IPv6 ICMP Echo-Request?"; itype:128; 
   classtype:icmp-event; sid:2000001; rev:1;)
```

Good for application layer checks
Bad for protocol layer detection

⇒ need to develop a IPv6-Plugin
Snort Customizations

- Writing rules
- Dynamic Detection API: compiled rule evaluations
- Dynamic Preprocessor API:
  - add rule options
  - do something with a packet
New IPv6 Rule Options

Goal: Provide IPv6 access for signatures

- Basic Header
- Extension Headers
- Neighbor Discovery Options

Functionality:

- Handler for option parsing on config (re-)load
- Callbacks for option keywords
- Called with rule parameter and current packet
- Return `match/no_match`
Implementation

// IPv6_Rule_Init() reads rule "ipv: 6;" into IPv6_RuleOpt_Data

int IPv6_Rule_Eval(void *raw_packet, const u_int8_t **cursor, void *data)
{
    SFSnortPacket *p = (SFSnortPacket*) raw_packet;
    struct IPv6_RuleOpt_Data *sdata = (struct IPv6_RuleOpt_Data *) data;

    switch (sdata->type) {
    case IPV6_RULETYPE_IPV: {
        uint_fast8_t ipv = GET_IPH_VER(p);
        if (checkField(sdata->op, ipv, sdata->opt.number))
            return RULE_MATCH;
        else
            return RULE_NOMATCH;
    // ...
    }
    }
}
IPv6 Rule Options

alert icmp any any -> any any (itype:8; ipv: 4; \
    msg:"ICMPv4 PING in v4 pkt"; sid:1000000; rev:1;)
alert icmp any any -> any any (itype:8; ipv: 6; \
    msg:"ICMPv4 PING in v6 pkt"; sid:1000001; rev:1;)

alert icmp any any -> any any (itype:128; ipv: 4; \
    msg:"ICMPv6 PING in v4 pkt"; sid:1000002; rev:1;)
alert icmp any any -> any any (itype:128; ipv: 6; \
    msg:"ICMPv6 PING in v6 pkt"; sid:1000003; rev:1;)

Martin Schütte
Resulting Evaluation Tree

Port Group
ICMP any->any

NC Rule
Tree Root

itype:8

itype:128

ipv:4

ipv:6

leaf

leaf
# Rule Options of the IPv6-Plugin

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ipv</td>
<td>IP version</td>
</tr>
<tr>
<td>ip6_tclass</td>
<td>Traffic Class</td>
</tr>
<tr>
<td>ip6_flow</td>
<td>Flow Label</td>
</tr>
<tr>
<td>ip6_exthdr</td>
<td>Extension Header</td>
</tr>
<tr>
<td>ip6_extnum</td>
<td>Num. of Ext Hdrs.</td>
</tr>
<tr>
<td>ip6_ext_ordered</td>
<td>Ext Hdrs. correctly ordered (bool)</td>
</tr>
<tr>
<td>ip6_option</td>
<td>Destination-/HbH-Option</td>
</tr>
<tr>
<td>ip6_optval</td>
<td>Destination-/HbH-Option Value</td>
</tr>
<tr>
<td>ip6_rh</td>
<td>Routing Header</td>
</tr>
<tr>
<td>icmp6_nd</td>
<td>Neighbor Discovery (bool)</td>
</tr>
<tr>
<td>icmp6_nd_option</td>
<td>Neighbor Discovery Option</td>
</tr>
</tbody>
</table>

(Most rules accept comparison operators = ! < >)
More Examples

alert ip any any -> any any (ip6_rh: !2;
  msg:"invalid routing hdr";
  sid:1000004; rev:1;)

alert ip any any -> any any (ip6_option: 0.0xc2;
  msg:"ip6 option: Jumbo in HBH hdr";
  sid:100066; rev:1;)

# event threshold
alert icmp any any -> any any (icmp6_nd;
  detection_filter: track by_dst, count 50, seconds 1;
  msg:"ICMPv6 flooding";
  sid:100204; rev:1;)

# log only one flooding event per second:
event_filter gen_id 1, sig_id 100204,
  type limit, track by_src,
  count 1, seconds 1
Preprocessor for Neighbor Discovery Tracking

Goal: monitor network changes

- new hosts
- new routers
- basic extensions/options check

Functionality:

- Reads ICMPv6 messages
- Follows network state, i.e. (MAC, IP) tuple of:
  - On-link Routers
  - On-link Hosts
  - Ongoing DADs
- Alert on change
Configuration

in snort.conf, all optional

- net_prefix: subnet prefixes
- router_mac: known router MAC addresses
- host_mac: known host MAC addresses
- max_routers: max routers in state (default: 32)
- max_hosts: max hosts in state (default: 8K)
- max_unconfirmed: max unconfirmed nodes in state (default: 32K)
- keep_state: remember nodes for \( n \) minutes (default: 180)
- expire_run: clean memory every \( n \) minutes (default: 20)
- disable_tracking: only rules & stateless checks (default: false)
Configuration

“normal use”

preprocessor ipv6:
  net_prefix 2001:0db8:1::/64
  router_mac 00:16:76:07:bc:92
Preprocessor State at Runtime

```c
struct IPv6_State
{
    struct IPv6_Hosts_head *routers;
    struct IPv6_Hosts_head *hosts;
    struct IPv6_Hosts_head *unconfirmed;
    struct IPv6_Statistics *stat;
    struct IPv6_Config *config;
    time_t next_expire;
}

struct IPv6_Hosts_head
{
    struct RB_HEAD(IPv6_Host) data;
    u_int32_t entry_limit;
    u_int32_t entry_counter;
}

struct IPv6_Host
{
    RB_ENTRY(IPv6_Host) entries;
    u_int8_t ether_source[6];
    sfip_t ip;
    ...
}

struct IPv6_Statistics
{
    uint32_t pkt_seen;
    uint32_t pkt_fragments;
    uint32_t pkt_icmpv6;
    ...
}

struct IPv6_Config
{
    u_int32_t max_routers;
    u_int32_t max_hosts;
    u_int32_t max_unconfirmed;
    struct MAC_Entry_head *router_whitelist;
    struct MAC_Entry_head *host_whitelist;
    ...
}
```
Implementation of NS Processing

```c
static void IPv6_Process_ICMPv6_NS(const SFSnortPacket *p, struct IPv6_State *context) {
    struct nd_neighbor_solicit *ns = (struct nd_neighbor_solicit *) p->ip_payload;
    sfip_t *target_ip;
    struct IPv6_Host *ip_entry;

    target_ip = sfip_alloc_raw(&ns->nd_ns_target, AF_INET6, &rc);
    // ..

    ip_entry = get_host_entry(context->hosts, target_ip);
    if (ip_entry) {
        DEBUG_WRAP(DebugMessage(DEBUG_PLUGIN, "Neighbour solicitation from known host\n");)
        return;
    }

    /* this is the expected part: the IP is yet unknown --> put into DAD state */
    ip_entry = create_dad_entry_ifnew(context->unconfirmed,
        &p->pkt_header->ts,
        p->ether_header->ether_source,
        target_ip);

    if (!ip_entry) {
        DEBUG_WRAP(DebugMessage(DEBUG_PLUGIN, "create_dad_entry_ifnew failed\n");)
        return;
    }

    DEBUG_WRAP(DebugMessage(DEBUG_PLUGIN, "%s DAD started by %s / %s\n",
        pprint_ts(ip_entry->last_adv_ts),
        pprint_mac(ip_entry->ether_source),
        sfip_to_str(&ip_entry)););

    _dpd.alertAdd(GEN_ID_IPv6, SID_ICMP6_ND_NEW_DAD, 1, 0, 3, SID_ICMP6_ND_NEW_DAD_TEXT, 0 );
}
```
## Snort IPv6 Alerts: ND Tracking

<table>
<thead>
<tr>
<th>SID</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>RA from new router</td>
</tr>
<tr>
<td>2</td>
<td>RA from non-router MAC address</td>
</tr>
<tr>
<td>3</td>
<td>RA prefix changed</td>
</tr>
<tr>
<td>4</td>
<td>RA flags changed</td>
</tr>
<tr>
<td>5</td>
<td>RA for non-local net prefix</td>
</tr>
<tr>
<td>6</td>
<td>RA with lifetime 0</td>
</tr>
<tr>
<td>7</td>
<td>new DAD started</td>
</tr>
<tr>
<td>8</td>
<td>new host in network</td>
</tr>
<tr>
<td>9</td>
<td>new host with non-allowed MAC addr.</td>
</tr>
<tr>
<td>10</td>
<td>DAD with collision</td>
</tr>
<tr>
<td>11</td>
<td>DAD with spoofed collision</td>
</tr>
</tbody>
</table>
## Snort IPv6 Alerts: Packet Attributes

<table>
<thead>
<tr>
<th>SID</th>
<th>Message</th>
</tr>
</thead>
<tbody>
<tr>
<td>12</td>
<td>mismatch in MAC/NDP src ll addr.</td>
</tr>
<tr>
<td>13</td>
<td>extension header has only padding</td>
</tr>
<tr>
<td>14</td>
<td>option lengths $\neq$ ext length</td>
</tr>
<tr>
<td>15</td>
<td>padding option data $\neq$ zero</td>
</tr>
<tr>
<td>16</td>
<td>consecutive padding options</td>
</tr>
</tbody>
</table>
**Extremely** useful for development.

Verify intended results for given packet samples.
Output/Visualization

- Big Problem
- `barnyard2` tool for Snort log processing (e.g. write SQL)
- Few Open Source frontends (BASE & Snorby)
- All using old SQL Schema, without IPv6 field
Performance

Theory:
- Stateless checks require processing
- ND Tracking requires memory $\Rightarrow$ DoS risk

Practice:
- Snort’s packet decoding does 90% of the work
- Configurable memory limit $\sim$ 8 Mb
- TCP stream reassembly is much more expensive
Bugs Found in Snort

or: Real-World Problems of Major Commercial Security Products

- Ping of Death, cannot process > 40 extension headers
- wrong Endianness in GET_IPH_VER()
- fragmentation breaks ICMP/UDP checksums
- Routing Headers break ICMP/UDP checksums
- fragbits rules not supported
Extension Header Parsing in Snort 2.9.0

void DecodeIPV6Options(int type, const uint8_t *pkt, uint32_t len, Packet *p)
{
    uint32_t hdrlen = 0;

    if(p->ip6_extension_count < IP6_EXTMAX) {
        switch (type) {
            case IPPROTO_HOPOPTS:
                hdrlen = sizeof(IP6Extension) + (exthdr->ip6e_len << 3);
        }
    }
    /* missing else => hdrlen=0 => infinite mutual recursion */

    DecodeIPV6Extensions(*pkt, pkt + hdrlen, len - hdrlen, p);
}

void DecodeIPV6Extensions(uint8_t next, const uint8_t *pkt, uint32_t len, Packet *p)
{
    switch(next) {
        case IPPROTO_HOPOPTS:
        case IPPROTO_DSTOPTS:
        case IPPROTO_ROUTING:
        case IPPROTO_AH:
            DecodeIPV6Options(next, pkt, len, p);
            return;
    }
}
Conclusion

• It works!

• Dynamic Library (no need to recompile Snort)
• Enables IPv6-specific detection signatures
• Snort & IPv6-Plugin detects THC attacks

• Cannot solve fundamental problems: DoS and insecure Ethernet
• Can raise visibility and awareness of network threat situation
Contact

E-Mail: info@mschuette.name
Project Page: http://mschuette.name/wp/snortipv6/
Source Code: https://github.com/mschuett/spp_ipv6