IPv6 Intrusion Detection mit Snort-Plugin

Martin Schütte

Problem

- IPv6 kommt ins Netz
- wenig Erfahrung mit Protokoll-Sicherheit
- bekannte Design- & Implementierungs-Fehler

Problem

- IPv6 kommt ins Netz
- wenig Erfahrung mit Protokoll-Sicherheit
- bekannte Design- & Implementierungs-Fehler

Ziel: Erkenne Angriffe mit Intrusion Detection System.

Angriffe auf IPv6

Das übliche:

- Wertebereiche für Felder
- Fragmentierung
- Denial of Service
- Portscans
- Fehler in Anwendungsschicht

IPv6-spezifisch:

- variable Header
- Autoconfiguration
- Routing
- Multicast

Angriffe auf IPv6

Das übliche:

- Wertebereiche für Felder
- Fragmentierung
- Denial of Service
- Portscans
- Fehler in Anwendungsschicht

IPv6-spezifisch:

- variable Header
- Autoconfiguration
- Routing
- Multicast

RFCs von 1995/1998 $\Rightarrow$ 15 Jahre Sicherheits-Erfahrung nachzuholen

Prämisse: lokales Netz ist sicher und vertrauenswürdig.

Beispiel-Angriff

einfacher Denial of Service:

1. Host Alice startet *Duplicate Address Detection*
   „Benutzt jemand die IP *X*?“
2. Host Eve antwortet „Ich benutze IP *X*.“
3. `goto 1`

Routing/Man in the Middle:

1. Host Eve sendet ICMPv6 Redirect
   „Hier Router Bob, für *google.com* bitte Router Eve benutzen.“

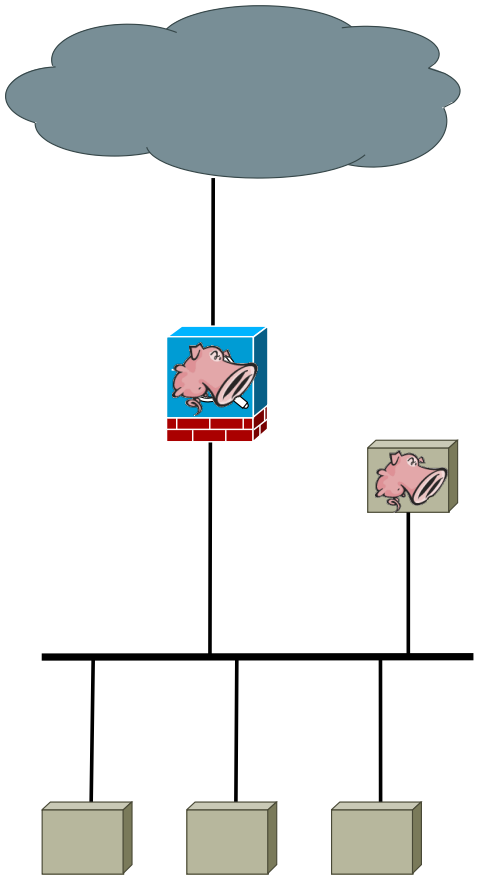
THC Toolkit

Zielt auf:

- Autoconfiguration (`dos-new-ip6`)
- Neighbour Cache (`parasite6`)
- Routing (`fake_router6`, `redir6`)
- DoS (`flood_router6`, `flood_advertise6`, `toobig6`, `smurf6`, `rsmurf6`, `sendpees6`)

Zielsystem: Snort 2.9.x

- verbreitetes Open Source NIDS
- Packet-Sniffer (IDS)
- Filter-/inline-Modus (IPS)
- Dekoder für gängige IPv6-Tunnelprotokolle



IPv6 Support

im Prinzip Ja, aber . . .

Alle relevanten IDS haben IPv6-Support.

Aber was heißt das?

- Fragment-Reassemblierung
- TCP & UDP Dekodierung

Bisher keine Erkennung IPv6-spezifischer Angriffe.

IPv6 Signaturen

Protokollfelder und Regeln abwärtskompatibel.

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

Aber keine Schlüsselwörter für neue IPv6-Felder.
⇒ Kein einfacher Weg IPv6-spezifische Regeln zu schreiben.

IPv6 Signaturen

Protokollfelder und Regeln abwärtskompatibel.

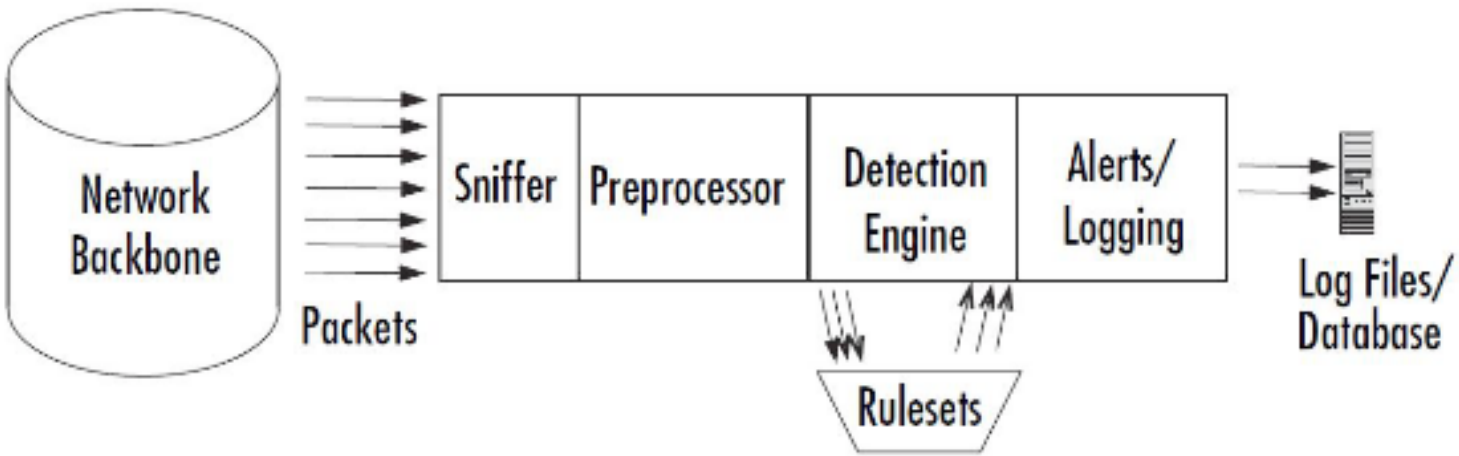
```
alert ip icmp any -> any any                                    \
    (msg:"IPv6 ICMP Echo-Request"; itype:128;                   \
    classtype:icmp-event; sid:2000001; rev:1;)
```

Aber keine Schlüsselwörter für neue IPv6-Felder.

⇒ Kein einfacher Weg IPv6-spezifische Regeln zu schreiben.

⇒ Eigenes Plugin dafür!

Kleines Snort-Plugin How-To



Plugin-APIs für Präprozessoren und Regeln.

Kleines Snort-Plugin How-To

dynamische Präprozessoren und Regeln

Was ist ein Plugin?

- dynamische Bibliothek (`.so`)
- aus `snort.conf` geladen, aktiviert, konfiguriert
- Aufruf über zwei Symbole:
  `LibVersion` und `InitializePreprocessor`
- Snort stellt ≈ 80 Funktionen bereit
- eigene Funktionen über Callback-Registrierung
  (Aktivierung, Paket-Verarbeitung, Reload, etc.)

Plugin registrieren

dynamicpreprocessor file /lib/libsf_ipv6_preproc.so

```
#define DYNAMIC_PREPROC_SETUP    IPv6_Preproc_Setup

PREPROC_LINKAGE int InitializePreprocessor(
    DynamicPreprocessorData *dpd)
{
    _dpd = *dpd;
    DYNAMIC_PREPROC_SETUP();
    return 0;
}

void IPv6_Preproc_Setup(void)
{
  _dpd.registerPreproc("ipv6", IPv6_Init);
}
```

Plugin aktivieren

preprocessor ipv6: option123

```
static void IPv6_Init(char *args)
{
   config = malloc(...);
   Parse_Config(args, config);

   _dpd.addPreproc(IPv6_Process,          // callback
       PRIORITY_NETWORK,                  // layer
       PP_IPv6,                           // set ID
       PROTO_BIT__ALL);                   // L4 protocol

   _dpd.registerPreprocStats("ipv6", IPv6_PrintStats);
}
```

Paket verarbeiten

```
void IPv6_Process(void *pkt, void *snortcontext)
{
  SFSnortPacket *p = (SFSnortPacket *) pkt;

  context->stat->pkt_seen++;

  if (!p || !p->ip6h)
    return;

  switch (p->ip6h->next) {
  case IPPROTO_ICMPV6:
      IPv6_Process_ICMPv6(p, context);
      break;
  ...
```

Paket verarbeiten, forts.

```
// check for routing header
for(i = 0; i < p->num_ip6_extensions; i++) {
  if (option_type == IPPROTO_ROUTING) {
    rthdr = (struct ip6_rthdr *) option_data;

    if (rthdr->ip6r_type == 0)
      _dpd.alertAdd(GEN_ID_IPv6, SID_IP6_RH0,
                    1, 0, 3, "found RH0", 0 );

    if (rthdr->ip6r_type == 2
        && context->config->warn_on_mobile)
      _dpd.alertAdd(GEN_ID_IPv6, SID_MOBILE,
                    1, 0, 3, "found Mobile IP", 0 );
  }
}
```

IPv6 Präprozessor

Funktionsweise:

- Prüft Protokollfelder
- Zählt Autoconf-Nachrichten (DoS-Erkennung)
- Verfolgt Netz-Zustand, d. h. MACs & IPs von Routern, Hosts, laufenden DADs

Grundlegende Einschränkung:

- Layer 2/Ethernet: keine sichere ID

IPv6 Checks

derzeit vorhandene SIDs

- Ungültige/ungewöhnliche Werte
  - Hop-by-Hop extension is not first in packet
  - Too many extension headers
  - Ping from/to IPv6 multicast address
  - Packet with RFC3692-style experimental values
  - Packet with unassigned values
  - Packet with deprecated RH0 header (source routing)

IPv6 Checks

derzeit vorhandene SIDs

- Route-Änderungen
  - RA from new router
  - RA prefix changed
  - RA flags changed
- Autokonfiguration
  - ND from new host in network
  - DoS on autoconf

IPv6 Checks

derzeit vorhandene SIDs

- ggf. unbenutzte Protokolle
  - DHCP6 advertisement
  - IPSec packet
  - Mobile IPv6
  - Redirect packet
  - SEND packet

Konfiguration

in `snort.conf`

```
preprocessor ipv6:                                  \
  net_prefix 2001:0db8:1::/64                       \
  router_mac 00:16:76:07:bc:92                      \
  host_mac                                          \
  max_extensions          8                         \
  max_hosts           16384                         \
  max_dads            16384                         \
  threshold_ra            8                         \
  threshold_na            8                         \
  expect_send           expect_mobile               \
  expect_ipsec          expect_dhcp                 \
  expect_unassigned     expect_experimental
```

Performance

Zwei Komponenten

- Zustandslose Checks sind schnell:
  Decoder eledigt die Arbeit; Plugin vergleicht nur noch Protokoll-Felder in `struct SFSnortPacket`
- Zustand verfolgen kostet Zeit und Speicher:
  Knoten suchen, Speicher belegen, Werte kopieren, in Baum einfügen, ... $\Rightarrow$ DoS-Gefahr

Test

Einige Tests für bestimmte Alarm-Bedingungen:

- PCAP-Datei als Eingabe
- optional: `snort.conf`-Fragment
- Alarm-Spezifikation (erwartete SIDs)

Bisher Snort-spezifisch, aber vielleicht Basis für IDS-Vergleiche.

Snort-Funktionen

IPv6-fertige Snort-Komponenten

- Portscans (`sfportscan`) & Fragmentierung (`frag3`)
- Paketfilter (Inline-Modus, je nach DAQ)
- Logging (`unified2`)

(größtes?) Problem: kein SQL-Schema für IPv6-Events
⇒ keine IPv6-Events in barnyard2-DB-Output, BASE, Snorby, u. ä.

Prüfung weiterer Protokollschichten

- Multicast Listener Discovery (MLD)
- DHCPv6
- Secure Neighbor Discovery (SEND)
- IPSec
- Mobile IPv6
- ...

Wo gibt's das Plugin?

Noch nicht veröffentlicht, da Teil laufender Diplomarbeit.

Bei Interesse bitte E-Mail schreiben (`info@mschuette.name`).

Nützliche Programme

THC	http://freeworld.thc.org/thc-ipv6/
Scapy	http://www.secdev.org/projects/scapy/
Wireshark	http://www.wireshark.org/
ndpmon	http://ndpmon.sf.net/
Snort	http://www.snort.org/
Suricata	http://www.openinfosecfoundation.org/
SnortUnified.pm	http://code.google.com/p/snort-unified-perl/