BEUTH HOCHSCHULE FÜR TECHNIK BERLIN
University of Applied Sciences

IPv6 Intrusion Detection mit Snort

Thomas Scheffler / Edurne Izaguirre (Beuth Hochschule)
Bettina Schnor / Martin Schütte (Universität Potsdam)

BLIT2010 - Potsdam, 6. November 2010

Übersicht

- Kurzvorstellung Snort
- 1x1 der Snort Regeln
- Installation eine IPv6-fähigen Snort Systems
- Was funktioniert? Was funktioniert nicht?
- Erkennungsszenarien
- Fazit

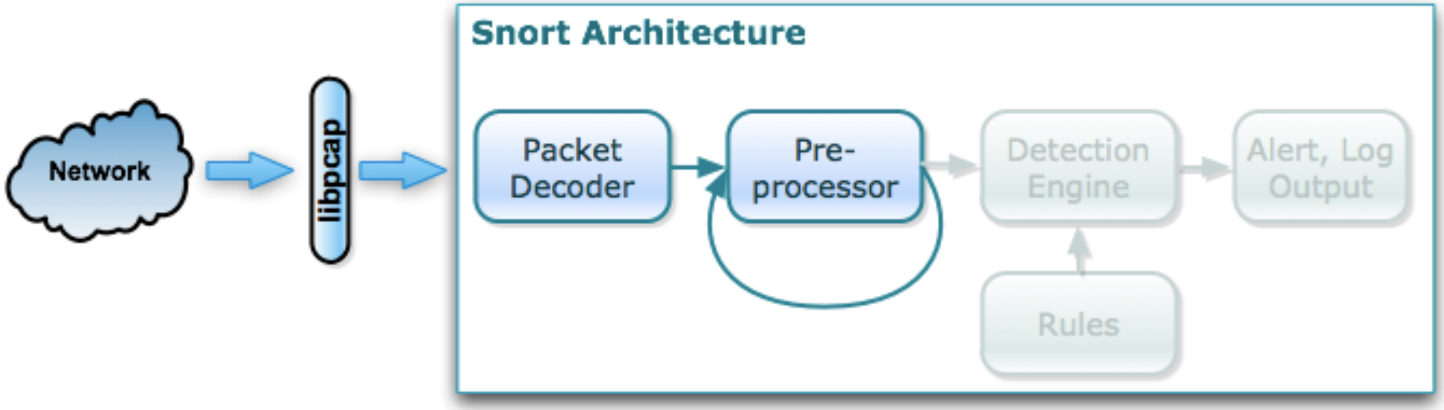
Snort - Kurzvorstellung

Snort - Kurzvorstellung

- **Snort**
  - Bekanntes und leistungsfähiges Open-Source Intrusion Detection und Preventionsystem (IDS/IPS)
  - Gestartet als ‘Cross-platform’ Sniffer 1998 von Martin Roesch
  - Signatur-basierte IDS
  - Januar 2001, Roesch gründet Sourcefire, Inc. um Snort auf eine kommerzielle Basis zu stellen und Support zu bieten
  - Snort 2.0 – neue Detection Engine für Gigabit-Netzwerke (2002)
  - Experimenteller IPv6-Support (Dezember 2002)
    - Kann nur Pakete dekodieren, keine Regelverarbeitung
  - Offizieller IPv6 Support mit Snort 2.8 (September 2007)
    - Seit Snort 2.8.4. werden alle Anwendungs-Präprozessoren unterstützt

Snort Architektur

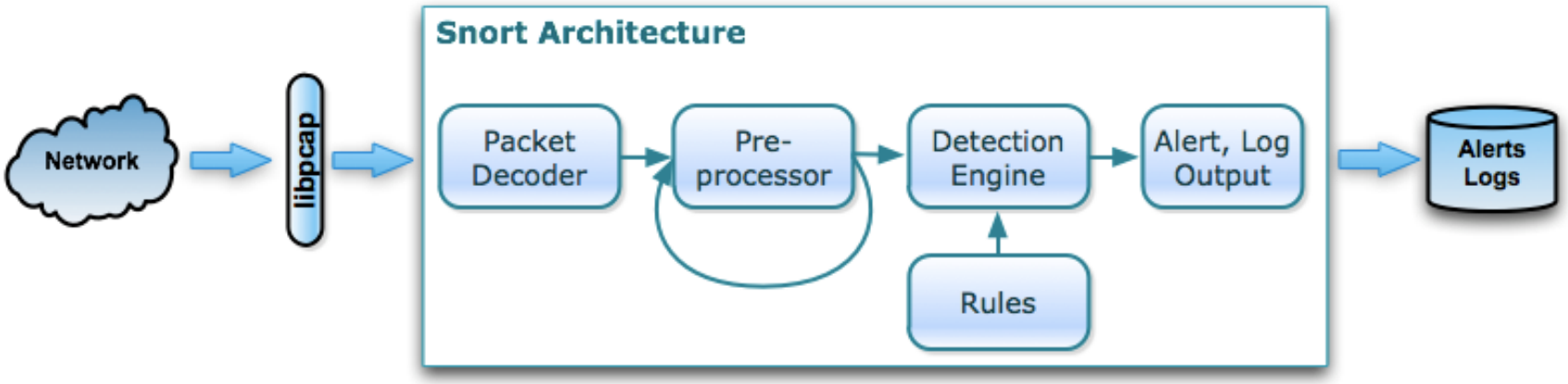
- **Packet Decoder**
  - Pakete werden durch die pcap-Bibliothek vom Netz gelesen
  - Der Decoder initialisiert und schreibt die Paket-Datenstruktur
- **Präprozessoren**
  - Untersuchen die Paket-Datenstruktur nach Auffälligkeiten (kompletter Zugriff auf alle Paketdaten möglich / Analysen können zustandsbehaftet sein)
  - Modifizieren die Datenstruktur (Normalisierung/Defragmentierung des Datenstroms)
  - Erleichtert die Anwendung von Content-Regeln



Snort Architektur

- **Detection Engine**
  - Prüft jedes eingehende Paket gegen die Regeln, die in der Snort-Konfiguration aktiviert wurden (prüft Paket-Payload)

- **Logging, Alerting und Output Modul**
  - Es existieren verschiedene Ausgabeoptionen (Log, Alert, Drop)
    - Unified Logging für hohe Geschwindigkeit
    - Direkter Support für verschiedene Datenbanken



Kleines 1x1 der Snort-Regeln

Eine kurze Übersicht der Snort Rules Language

Regeln bestehen aus zwei Sektionen: **Header** und **Options**

HEADER:

- Festlegen der Aktion: alert, log (drop – im Inline-Modus)
- Protokoll: IP, TCP, UDP and ICMP (keine IPv6 Spezifika möglich)
- Quell- und Ziel-IP Adresse
  - ✓ 192.168.1.0/24, FD00:141:64:1::/64
- Quell-Ports und Ziel-Ports
  - ✓ Einzelne Ports: 21
  - ✓ Festlegen von Bereichen: 1:1024 möglich
- Aus Gründen der Übersichtlichkeit und Weiterverwendbarkeit sollten Variablen genutzt werden

```
alert ip $HOME_NET 23 -> fd00:141:64:1::1 any
```

Snort Regeln: Options

OPTIONS:

- Optionen werden durch eine Klammer vom Regel-Kopf getrennt
- Die einzelnen Optionen werden durch ein Semikolon abgeschlossen(**;**)
- Schlüsselworte werden durch den Doppelpunkt(**:**) kennzeichnet

Kategorien:

- General, inklusive Meta-data und verschiedener Rule Optionen
- Payload
- Non-payload

Snort Rules: General Options

msg

- Angabe der Nachricht, die als Alert ausgegeben wird

sid

- Eindeutiger Identifikator für Snort Regeln

< 100	reserviert
100 – 1.000.000	Offizielle Snort-Regeln
> 1.000.000	Lokale, benutzerdefinierte Regeln

classtype

- Snort nutzt eine Klassifikationstabelle für Regeln, die gleichzeitig auch die Schwere des Angriffs festlegt

```
alert ip any any -> any any (msg:"Any IP Packet"; sid:2000000;)
```

Snort Regel: Payload Options

- **content**
  - sucht nach einem speziellen Muster in der Paket-Payload
  - Die Regel triggert wenn das Muster gefunden wird
- **nocase**
  - die Suche ist unabhängig von Groß/Kleinschreibung
- **depth**
  - Snort soll die ersten *x*-bytes der Payload nach dem Muster durchsuchen

```
alert tcp any 80 -> any any (msg:"4chan traffic"; content:"4chan"; \
        classtype:string-detect;)
```

```
alert tcp any any -> any 25 (msg:"SMTP expn root"; flags:A+; \
        content:"expn root"; nocase; classtype:attempted-recon;)
```

Snort Regeln: Non-Payload Options

itype

- Überprüft den Wert des ICMP-Type Felds

icode

- Überprüft den Wert des ICMP-Code Felds

flags

- Überprüft die TCP-Flags
- Kann durch +,*,! modifiziert werden

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

Installation eines IPv6-fähigen Snort-Systems

Testbed-Architektur

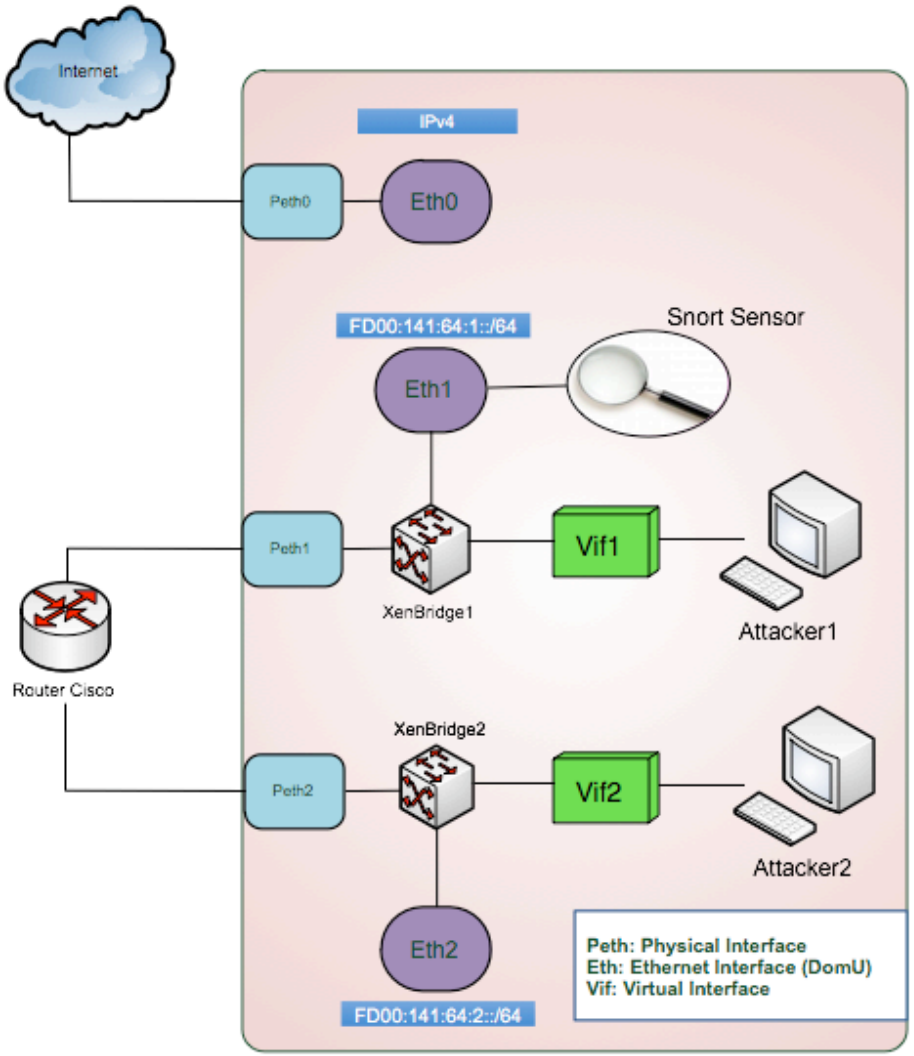
Virtualisiertes IPv6 Testbed

- Xen-basiert
- IPv6 Routing über Cisco2600

- Attacker1
  - CentOS Gast
- Attacker2
  - CentOS Gast

Snort 2.8.5.3

- in Dom0 installiert
- flexible Anordnung der Sensoren möglich



Installation

Installation des Systems

- Konfigurationsoptionen für IPv6 Support:

```
$ ./configure --enable-dynamicplugin \
              --enable-decoder-preprocessor-rules \
              --enable-ipv6
$ make
$ make install
```

Optional:

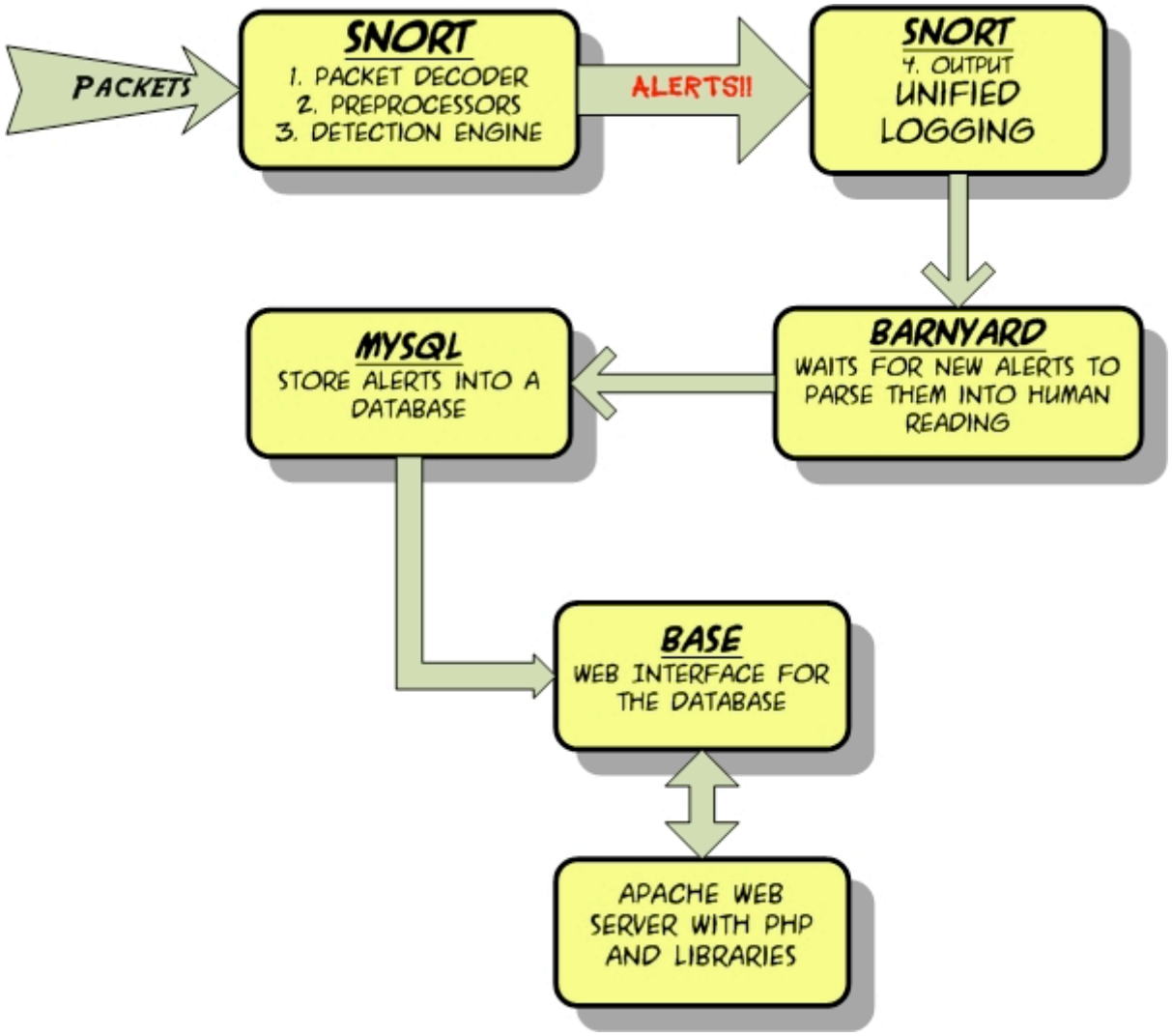
```
--with-mysql --with-mysql-libraries=/usr/lib64/mysql
```

```
$ snort --version

  ,,_     -*> Snort! <*-
 o"  )~   Version 2.8.5.3 IPv6 (Build 124)
  ''''    By Martin Roesch & The Snort Team:
          http://www.snort.org/snort/snort-team

          Copyright (C) 1998-2009 Sourcefire, Inc., et al.
          Using PCRE version: 6.6 06-Feb-2006
```

Empfohlener Aufbau für ein flexibles Snort System



Installation

Anpassung der Snort Konfigurationsdatei (snort.conf):

- **Festlegung des Output-formats und Orts**
  - Datenbank (MySQL, …)
  - Barnyard (Unified Logging nach /var/log/snort/)

- **Anpassung von Variablen**
  - HOME_NET: IPv4 und IPv6
  - EXTERNAL_NET: IPv4 und IPv6

```
IPVAR HOME_NET [10.0.1.0/24,FD00:141:64:1::/64]
```

- **Anpassung der Regelbasis**
  - Welche Regeln (Regelgruppen) soll Snort auswerten?

Was funktioniert in Snort für IPv6?

Was funktioniert mit Snort IPv6?

- **In Snort Rules**
  - Es gibt keine speziellen Schlüsselwort für die neuen Felder im IPv6 Header
  - Die *ip_proto* Schlüsselwörter (**ip**, **icmp**, **tcp**, **udp**) machen keine Unterscheidung zwischen den zwei IP-Protokollversionen
  - Adressvariablen müssen den neueren Variablentyp **ipvar** nutzen, Listen können gemischte Einträge enthalten:
    - ✓ IPv4 => **var** 192.168.1.10/32
    - ✓ IPv6 => **ipvar** ff02::01

```
alert icmp 10.0.1.1 any -> any any (msg:"PING"; \
        itype:8; sid 2000003; rev:1;)
```

```
alert icmp fd00:141:64:2::1 any -> any any (msg: "PINGv6";\
        itype:128; sid:2000004; rev:1;)
```

Was funktioniert mit Snort IPv6

Präprozessoren mit IPv6 Support (seit Snort 2.8.4)

- Stream5
- HTTP Inspect
- DCERPC
- Portscan
- BO
- RPC Decode
- Frag3
- FTP Telnet
- DNS
- SMTP

IPv6 Support in der Regelbasis

- Normale Content-Regeln funktionieren automatisch auch für IPv6 Transport
- Dedizierte IPv6-Regeln sind in der Standard-Regelbasis sowie den Paket-Decoder-Regeln enthalten
- Teredo-Erkennung

Was funktioniert (noch) nicht?

Was funktioniert noch nicht?

MySQL Ausgabe

- IPv4 Adressen werden in der Datenbank mit dem Typ unsigned int (32 bit) gespeichert.
  => Feldlänge ist für IPv6 Adressen nicht ausreichend
- Events werden aber trotzdem in der Datenbank gespeichert

BASE (Basic Analysis and Security Engine)

- Unterstützt ebenso wie die Datenbank keine IPv6 Adressen

Prelude Ausgabe

- Plugin unterstützt kein IPv6
- Wenn `--enable-ipv6` und `--enable-prelude` gleichzeitig als Konfigurationsoptionen angegeben werden, dann erfolgt ein Silent-Fallback auf IPv4

Was funktioniert noch nicht?

Basic Analysis and Security Engine (BASE)

Home | Search

[Back]

Queried on : Thu May 13, 2010 18:07:29

Meta Criteria	*any*
IP Criteria	*any*
Layer 4 Criteria	*none*
Payload Criteria	*any*

Summary Statistics

- Sensors
- Unique Alerts
- (classifications)
- Unique addresses: Source | Destination
- Unique IP links
- Source Port: TCP | UDP
- Destination Port: TCP | UDP
- Time profile of alerts

Displaying 15 Last Alerts

ID	< Signature >	< Timestamp >	< Source Address >	< Dest. Address >	< Layer 4 Proto >
#0-(4-53)	[snort] IPv6 ICMP Echo-Request	2010-04-28 16:44:31	0.100.0.1	0.0.0.0	IP
#1-(4-52)	[snort] v6 Routing Header	2010-04-28 16:44:31	0.100.0.1	0.0.0.0	IP
#2-(4-50)	[snort] v6 Routing Header	2010-04-28 16:38:35	0.100.0.1	0.0.0.0	IP
#3-(4-51)	[snort] IPv6 ICMP Echo-Request	2010-04-28 16:38:35	0.100.0.1	0.0.0.0	IP
#4-(4-48)	[snort] v6 Routing Header	2010-04-28 16:34:48	0.100.0.1	0.0.0.0	IP
#5-(4-49)	[snort] IPv6 ICMP Echo-Request	2010-04-28 16:34:48	0.100.0.1	0.0.0.0	IP
#6-(4-47)	[snort] IPv6 ICMP Echo-Request	2010-04-28 16:17:30	0.100.0.1	0.0.0.0	IP
#7-(4-46)	[snort] IPv6 ICMP Echo-Request	2010-04-28 16:06:48	0.100.0.1	0.0.0.0	IP
#8-(4-45)	[snort] IPv6 ICMP Echo-Request	2010-04-28 16:01:37	0.100.0.1	0.0.0.0	IP

Was funktioniert noch nicht?

BASE (Detailansicht)

Meta				
	ID #	**Time**	**Triggered Signature**	
	4 - 53	2010-04-28 16:44:31	[snort] IPv6 ICMP Echo-Request	
Sensor	**Sensor Address**	**Interface**	**Filter**	
	2	eth1	*none*	
Alert Group	*none*			

IP											
Source Address	**Dest. Address**	**Ver**	**Hdr Len**	**TOS**	**length**	**ID**	**fragment**	**offset**	**TTL**	**chksum**	
0.100.0.1	0.0.0.0	6	0	0	0	32	no	0	253	321 = 0x141	
Options	*none*										

Payload	
Normal Display	
Download of Payload	*none*
Download in pcap format	

Was funktioniert noch nicht?

Sonstiges:

- Die meisten Präprozessoren werden für IPv6 unterstützt.
  - Für einige spezielle Plugins/Präprozessoren ist der Status der IPv6-Unterstützung ungeklärt.
- Derzeit existieren noch keine Regel-Erweiterungen für die Inspektion von IPv6 Erweiterungsheadern. In einem späteren Release soll dieses nachgeholt werden.
- Es gibt keine spezielle Unterscheidung für ICMP6; das Schlüsselworts ICMP triggert auf beide Protokollversionen.
  - Potentiell problematisch sind dabei ICMP-Typen 3 und 4, die für ICMPv4/v6 unterschiedliche Bedeutung haben.

Was funktioniert noch nicht?

Output Modul

- Prelude und Datenbanksupport für IPv6

Präprozessoren mit ungeklärtem IPv6 Support

- Aruba
- Respond / Respond2
- Dynamic plugins (Shared Object rules)
- Stream4 (statt dessen Stream 5 benutzen)
- Flow (statt dessen Stream 5 benutzen)

Wie testen wir das IPv6 IDS?

Wie testen wir das IPv6 IDS?

Tools:

- Scapy
  - Python-basiertes Paketcrafting-Tool
  - http://www.secdev.org/projects/scapy/
- Bekannte Test- und Angriffswerkzeuge
  - Nmap
  - THC-IPv6-Toolkit
  - ~~Nessus~~
- …

Szenarien

Erkennung von Routing Headern
Erkennung eines ‘Rough‘-Routers
Erkennung von IPv6-Portscans

Erkennung von Routing Headern

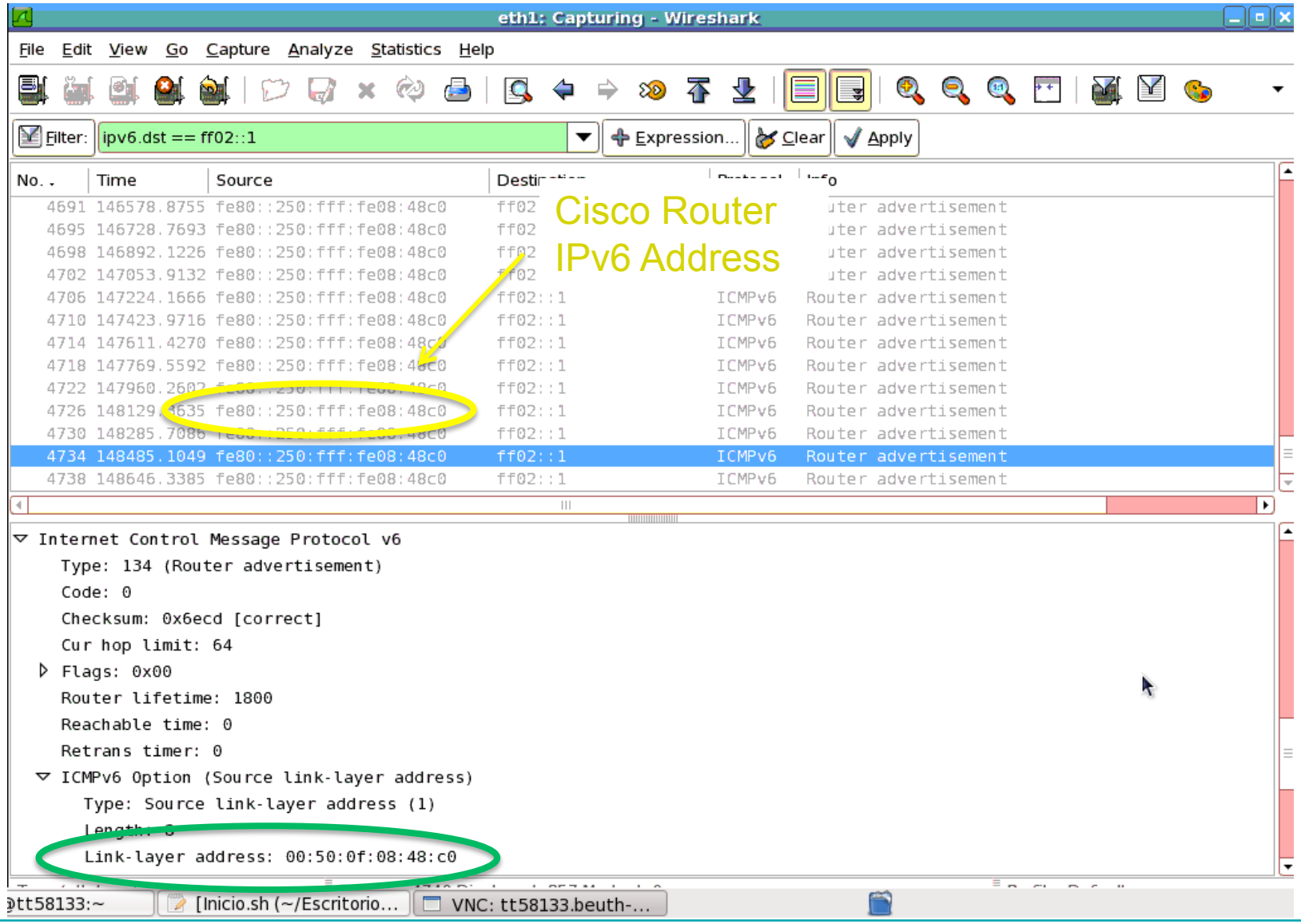
Erkennung des Next-Headers möglich (ip_proto:43)

- **aber:** derzeit kein direkter Zugriff auf Datenstrukturen in den Erweiterungs-Headern
- Snort-Regeln arbeiten auf L4-Ebene

- Protokoll-Anomalien werden durch den Packet-Decoder erkannt und gemeldet
  - Snort sollte dazu mit der folgenden Option übersetzt werden `--enable-decoder-preprocessor-rules`
  - Routingheader-0 soll in der nächsten Snort-Version erkannt werden

```
alert ip any any -> any any (ip_proto:43; msg:"IPv6 Routing Header
present"; classtype:attempted-recon; sid:2000006; rev:1;)
```

Eine Regel zur Erkennung von ‘Rough Routers’



Eine Regel zur Erkennung von ‘Rough Routers’

Wir schreiben unsere eigene Regel:

1. Ein korrektes Router-Advertisement wird analysiert und die wesentlichen Merkmale bestimmt:
   - MAC-Adresse des Routers
   - IPv6 Prefix in der ICMP-Option
2. Um flexible Regeln zu schreiben nutzen wir Variablen:
   ipvar ALL_NODES FF02::1

```
alert icmp any any ->  $ALL_NODES any  (msg:"Fake IPv6 Router
Advertisement"; itype:134; icode:0; content: !"|00 50 0f 08 48
c0|" ; offset:14; depth:20; content:!"|fd 00 01 41 00 64 00 01 00 00
00 00 00 00 00 00|"; distance:24;classtype:attempted-dos; sid:
2000007; rev:2;)
```

Erkennung von IPv6 Portscans

Portscan Präprozessor:

- Portscan-Präprozessor wird in `snort.conf` konfiguriert
- Test mit Nmap: `nmap -6 –sT`

Meta	ID #	Time	Triggered Signature
	4 - 56	2010-05-13 18:56:18	[snort] (portscan) TCP Portscan
	Sensor	Sensor Address	Interface	Filter
		2	eth1	*none*
	Alert Group	*none*		

IP	Source Address	Dest. Address	Ver	Hdr Len	TOS	length	ID	fragment	offset	TTL	chksum
	0.0.0.0	0.0.0.0	0	0	0	0	0	no	0	0	0 = 0x0
	Options	*none*									

Payload

Normal Display

Download of Payload

Download in pcap format

```
Priority Count: 5
Connection Count: 1826
IP Count: 1
Scanner IP Range: fd00:0001:0000:0000:0000:0000:0000:0001:fd00:0001:0000:0000:0000:0000:0000:0001
Port/Proto Count: 1823
Port/Proto Range: 1:65301
```

Fazit

Fazit

- Snort bietet integrierten IPv6 Support
  - Einfache Content-Regeln müssen für IPv6-Support nicht verändert werden!
- Prä- und Postprocessing ist noch verbesserungswürdig
  - Datenbankausgabe
  - Erkennung und Behandlung von Eigenarten des IPv6-Protokolls

Im Rahmen der gemeinsamen **IPv6-Arbeitsgruppe** der Beuth-Hochschule und des Instituts für Informatik der Uni-Potsdam wird derzeit ein **Snort-Präprozessor** für die Erkennung von Angriffen auf die Auto-Adresskonfiguration von IPv6 entwickelt.

Stay tuned!

Thomas Scheffler/ Edurne Izaguirre
Fachbereich Elektrotechnik
Beuth-Hochschule für Technik Berlin

Email: scheffler@beuth-hochschule.de
WWW: prof.beuth-hochschule.de/scheffler

Bettina Schnor / Martin Schütte
Institut für Informatik
Universität Potsdam

Email: schnor@cs.uni-potsdam.de
WWW: www.cs.uni-potsdam.de/bs