Capture Filter Sorcery
How to Use Complex BPF Capture Filters in Wireshark

June 15th, 2016

Sake Blok
sake.blok@SYN-bit.nl
Relational Therapist for Computer Systems | SYN-bit
About me…

SYN-bit
deep traffic analysis

#SharkFest16 • Computer History Museum • June 13-16, 2016
The history of BPF

The BSD Packet Filter: A New Architecture for User-level Packet Capture

James McCauley and van Jacobson
Lawrence Berkeley Laboratory
One Cyclotron Road
Berkeley, CA 94720

Abstract

Many commercial and open source packet sniffers today implement the use of general purpose workstations for network packet capture. This approach can be inefficient and may even be impractical in certain situations. A new architecture for network packet capture is described and evaluated. This architecture is the BSD Packet Filter (BPF). BPF is a new approach to network packet capture that substantially reduces the overhead of packet capturing.

1 Introduction

Capturing network packets is a well understood problem. Unfortunately, the variety of network protocols and media makes it impossible to get good work done. Although there are some interesting ideas, the sheer diversity of the problem makes it impossible to adapt these ideas to all situations. The BSD Packet Filter aims to address this problem by providing a new architecture that can capture network packets in a highly efficient manner. The BSD Packet Filter is a new architecture that can capture packets without the overhead of other packet capture tools. The BSD Packet Filter is designed to be highly efficient and to provide a low overhead method of capturing packets.

The BSD Packet Filter is a new architecture for network packet capture that substantially reduces the overhead of packet capturing. The BSD Packet Filter is a new architecture that can capture packets in a highly efficient manner.

https://youtu.be/XHlqIqPvKw8

#SharkFest16 • Computer History Museum • June 13-16, 2016
BPF virtual machine lives inside the kernel
Userspace program (like dumpcap) compiles a filter into BPF bytecode
Userspace program loads bytecode into the BPF vm
BPF vm uses bytecode to filter packets
No need to copy non-matching packets to user space
All jumps are forward to prevent kernel loops
Single BPF program max 4096 instructions
Extended use…

• Network/Packet related
  - iptables filtering rules (used by cloudflare for instance)
  - Network Function Virtualization (NFV)
  - Myself: DDoS protection on F5 for a customer

• (Linux) system related
  - Internal BPF
  - eBPF
Back to network packets…

• Why do you want to filter?
  - Overloading the CPU
  - Not enough disk I/O

• Why not?
  - You might miss important packets

• Best strategy:
  - Do not filter if you don’t need to
  - Use snaplength if you do not need payload
  - Filter out what you don’t want to see
  - Filter specifically for what you expect to see
Example of reducing packets

- `sctp.chunk_type == 4 || sctp.chunk_type==5` : 22089 frames

- `vrrp` : 13441 frames
  - `ether dst host 01:00:5e:00:00:12 or 33:33:00:00:00:12`

- `bfd` : 2284 frames

- `ospf` : 2251 frames
  - `ether dst host 01:00:5e:00:00:05`

- `arp` : 1960 frames
  - `arp or (vlan and arp) or (vlan and arp)`

- `slow` : 848 frames
  - `ether dst host 01:80:c2:00:00:02`

- `hsrp` : 192 frames
  - `ether dst host 01:00:5e:00:00:02`

- `stp` : 126 frames
  - `ether dst host 01:00:0c:cc:cc:cd or 01:80:c2:00:00:00`
Example of reducing packets

- 6910 packets kept out of 50101 (~86% reduction)
- 1.672.554 bytes kept out of 10.043.072 (~83% reduction)

Final filter:
- not (ether dst host 01:00:5e:00:00:12 or 33:33:00:00:00:12 or
  01:00:5e:00:00:05 or 01:80:c2:00:00:02 or 01:00:5e:00:00:02 or
  01:00:0c:cc:cc:cd or 01:80:c2:00:00:00 or arp or (vlan and (sctp[12]=4
  or sctp[12]=5 or arp)) or (vlan and (sctp[12]=4 or sctp[12]=5 or
  (udp dst port 3784 and udp[4:2]=32 and udp[11:1]=24) or arp)) )
Performance in post capture filtering

- display filters will first fully dissect a packet, then filter
- BPF filters will just look at specific offsets for specific values
  - first mismatch will make it go to the next packet
- using BPF in post capture processing not possible (yet?) in wireshark/tshark
- Use tcpdump or windump instead (for now)
BPF virtual machine

• Created with motorola 6502 in mind
  - Accumulator (A), index register (X)
  - Packet-based memory model
  - Direct/indirect addressing
  - Arithmetic and Conditional logic

• Bytecode
  - list of fixed size instructions
  - opcode (16 bits)
  - jump true (8 bits)
  - jump false (8 bits)
  - ‘k’ value (32 bits)

```plaintext
{ 0x28, 0, 0, 0x0000001c },
{ 0x15, 0, 14, 0x00000080 },
{ 0x20, 0, 0, 0x0000001a },
{ 0x15, 2, 0, 0xc0a80064 },
{ 0x20, 0, 0, 0x0000001e },
{ 0x15, 0, 10, 0xc0a80064 },
{ 0x30, 0, 0, 0x00000017 },
{ 0x15, 0, 8, 0x00000006 },
{ 0x28, 0, 0, 0x00000014 },
{ 0x15, 6, 0, 0x0000001ff },
{ 0xb1, 0, 0, 0x0000000e },
{ 0x48, 0, 0, 0x0000000e },
{ 0x15, 2, 0, 0x00000050 },
{ 0x48, 0, 0, 0x00000010 },
{ 0x15, 0, 1, 0x00000050 },
{ 0x6, 0, 0, 0x00000000 },
{ 0x6, 0, 0, 0x00000000 },
```
## Mnemonics

<table>
<thead>
<tr>
<th>Instruction</th>
<th>Addressing mode</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ld</td>
<td>1, 2, 3, 4</td>
<td>Load word into A</td>
</tr>
<tr>
<td>ldi</td>
<td>4</td>
<td>Load word into A</td>
</tr>
<tr>
<td>ldh</td>
<td>1, 2</td>
<td>Load half-word into A</td>
</tr>
<tr>
<td>ldb</td>
<td>1, 2</td>
<td>Load byte into A</td>
</tr>
<tr>
<td>ldx</td>
<td>3, 4, 5</td>
<td>Load word into X</td>
</tr>
<tr>
<td>ldxi</td>
<td>4</td>
<td>Load word into X</td>
</tr>
<tr>
<td>ldbx</td>
<td>5</td>
<td>Load byte into X</td>
</tr>
<tr>
<td>st</td>
<td>3</td>
<td>Copy A into M[]</td>
</tr>
<tr>
<td>stx</td>
<td>3</td>
<td>Copy X into M[]</td>
</tr>
<tr>
<td>jmp</td>
<td>6</td>
<td>Jump to label</td>
</tr>
<tr>
<td>ja</td>
<td>6</td>
<td>Jump to label</td>
</tr>
<tr>
<td>jeq</td>
<td>7, 8</td>
<td>Jump on k == A</td>
</tr>
<tr>
<td>jneq</td>
<td>8</td>
<td>Jump on k != A</td>
</tr>
<tr>
<td>jlt</td>
<td>8</td>
<td>Jump on k &lt; A</td>
</tr>
<tr>
<td>jle</td>
<td>8</td>
<td>Jump on k &lt;= A</td>
</tr>
<tr>
<td>jgt</td>
<td>7, 8</td>
<td>Jump on k &gt; A</td>
</tr>
<tr>
<td>jge</td>
<td>7, 8</td>
<td>Jump on k &gt;= A</td>
</tr>
<tr>
<td>jset</td>
<td>7, 8</td>
<td>Jump on k &amp; A</td>
</tr>
<tr>
<td>add</td>
<td>0, 4</td>
<td>A + &lt;x&gt;</td>
</tr>
<tr>
<td>sub</td>
<td>0, 4</td>
<td>A - &lt;x&gt;</td>
</tr>
<tr>
<td>mul</td>
<td>0, 4</td>
<td>A * &lt;x&gt;</td>
</tr>
<tr>
<td>div</td>
<td>0, 4</td>
<td>A / &lt;x&gt;</td>
</tr>
<tr>
<td>mod</td>
<td>0, 4</td>
<td>A % &lt;x&gt;</td>
</tr>
<tr>
<td>neg</td>
<td>0, 4</td>
<td>!A</td>
</tr>
<tr>
<td>and</td>
<td>0, 4</td>
<td>A &amp; &lt;x&gt;</td>
</tr>
<tr>
<td>or</td>
<td>0, 4</td>
<td>A</td>
</tr>
<tr>
<td>xor</td>
<td>0, 4</td>
<td>A ^ &lt;x&gt;</td>
</tr>
<tr>
<td>lsh</td>
<td>0, 4</td>
<td>A &lt;&lt; &lt;x&gt;</td>
</tr>
<tr>
<td>rsh</td>
<td>0, 4</td>
<td>A &gt;&gt; &lt;x&gt;</td>
</tr>
<tr>
<td>tax</td>
<td></td>
<td>Copy A into X</td>
</tr>
<tr>
<td>txa</td>
<td></td>
<td>Copy X into A</td>
</tr>
<tr>
<td>ret</td>
<td>4, 9</td>
<td>Return</td>
</tr>
</tbody>
</table>

### Addressing mode Syntax

<table>
<thead>
<tr>
<th>Syntax</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>x/%x</td>
<td>Register X</td>
</tr>
<tr>
<td>[k]</td>
<td>BBW at byte offset k in the packet</td>
</tr>
<tr>
<td>[x + k]</td>
<td>BBW at the offset X + k in the packet</td>
</tr>
<tr>
<td>M[k]</td>
<td>Word at offset k in M[]</td>
</tr>
<tr>
<td>#k</td>
<td>Literal value stored in k</td>
</tr>
<tr>
<td>4*([k] &amp; 0xf)</td>
<td>Lower nibble * 4 at byte offset k in the packet</td>
</tr>
<tr>
<td>L</td>
<td>Jump label L</td>
</tr>
<tr>
<td>#k,Lt,Lf</td>
<td>Jump to Lt if true, otherwise jump to Lf</td>
</tr>
<tr>
<td>#k,Lt</td>
<td>Jump to Lt if predicate is true</td>
</tr>
</tbody>
</table>

---

#SharkFest16 • Computer History Museum • June 13-16, 2016
BPF filter language

• The filter expression consists of one or more primitives.
• Primitives usually consist of an id preceded by one or more qualifiers.
• There are three different kinds of qualifier:
  - type qualifiers say what kind of thing the id name or number refers to.
  - dir qualifiers specify a particular transfer direction to and/or from id.
  - proto qualifiers restrict the match to a particular protocol.
• Use and, or and not to combine primitives and create complex filters.
• To save typing, identical qualifier lists can be omitted.
• Filter arithmetics

Grammar: the difference between knowing your shit and knowing you’re shit.

#SharkFest16 • Computer History Museum • June 13-16, 2016
Primitives

• Primitives usually consist of an id preceded by one or more qualifiers.
• There are three different kinds of qualifier:
  - type qualifiers say what kind of thing the id name or number refers to.
  - dir qualifiers specify a particular transfer direction to and/or from id.
  - proto qualifiers restrict the match to a particular protocol.

• Examples
  - host sharkfest.wireshark.org
  - src host 10.0.1.217
  - ether src host de:ad:be:ef:ca:fe
It's Logic

• Combine primitives to create complex filters
  - use ‘and’ or ‘&&’ to concatenate
  - use ‘or’ or ‘||’ to alternate
  - use ‘not’ or ‘!’ to negate
  - use parentheses to group

• Examples:
  - host 10.0.0.1 and host 10.0.0.2
  - host 10.0.0.1 or host 10.0.0.2
  - host 10.0.0.1 and host 10.1.1.1 and udp port 53 or
    host 10.1.2.1 and tcp port 80
  - host 10.0.0.1 and ((host 10.1.1.1 and udp port 53) or
    (host 10.1.2.1 and tcp port 80))
Omit duplicate qualifiers

• To save typing, identical qualifier lists can be omitted.

• Examples:
  - tcp port 80 or tcp port 8080
    ‣ tcp port 80 or 8080
  - host 10.0.0.1 and host 10.0.0.2
    ‣ host 10.0.0.1 and 10.0.0.2
  - src host 10.0.0.1 and dst host 10.0.0.2
    ‣ not identical qualifiers
Getting to specific bytes

• Use \( \ldots[x] \) to address certain byte
  - tcp[13] = 2
  - ip[8] = 64 or 128 or 255 does not work!

• Can get two bytes with \( \ldots[x:2] \) or four bytes with \( \ldots[x:4] \)
  - ether[0:4] = 0x109add00 and ether[4:2] = 0xd796
  - ether[0:6] = 0x109add00d796 is not allowed

Two bytes meet. First byte asks, "Are you ill?" Second byte replies, "No, just feeling a bit off."
Getting to specific bits

• Use the logical and (&) to extract one or more bits
  - eth[0:4] & 0xffffffff00 = 0x109add00

• Use the logical or (|) to combine bits

• Some fields have names for the offset and specific bits:
  - tcp[tcpflags] & (tcp-syn | tcp-ack) = tcp-syn
  - icmp[icmptype] = icmp-unreach

1 + 8 + 16 + 64 + 128 = 217
• Calculations, both on values as on indices
  - Standard: +, -, *, and /
  - Right shift: >>
  - Left shift: <<

• Examples:
  - tcp[((tcp[12:1] & 0xf0) >> 2):4] = 0x47455420
  - (ip[2:2] - ((ip[0]&0x0f)<<2) - ((tcp[12]&0xf0)>>2)) > 0
Offset alterations

• `vlan` keyword increases all offsets by 4
  - `vlan 10` and host `10.0.0.1`
  - `vlan 10` or `vlan 20` ==> unexpected result?
• `mpls` keyword increases all offsets by 4
• `pppoes` keyword increases all offsets by 8
Back to mnemonics

• Check whether a filter is correct
• Verify whether the filter will actually do what you intend
• Use wireshark’s “Compile BPF filter”
• Use ‘tcpdump -d’ (or ‘tcpdump -dd’ if you’re into bytecode ;-))
BPF machine code

```
$ tcpdump -d -s 128 "ip and host 1.1.1.1 and tcp port 80"
(000) ldh      [12]
(001) jeq      #0x800           jt  2 jf  16
(002) ld       [26]
(003) jeq      #0x1010101       jt  6 jf  4
(004) ld       [30]
(005) jeq      #0x1010101       jt  6 jf  16
(006) ldb      [23]
(007) jeq      #0x6             jt  8 jf  16
(008) ldh      [20]
(009) jset     #0x1fff          jt 16jf 10
(010) ldxb     4*([14]&0xf)
(011) ldh      [x + 14]
(012) jeq      #0x50            jt 15jf 13
(013) ldh      [x + 16]
(014) jeq      #0x50            jt 15jf 16
(015) ret      #128
(016) ret      #0
$
```
Check for ip

0000 00 12 1e bb d1 3b f8 1e df d8 87 48 08 00 45 00 .....;......H..E.
0010 02 bd 28 fb 40 00 40 06 fd 9f c0 a8 01 2b 01 01 ..(.@.......+Bf
0020 01 01 c3 f6 00 50 f1 b8 8d 85 db 07 fd 9e 80 18 .g...P........
0030 ff ff ce b2 00 00 01 01 08 0a 2e b9 c5 24 03 63 ............$..c
0040 c5 41 47 45 54 20 2f 20 48 54 50 2f 31 2e 31 .AGET / HTTP/1.1
0050 0d 0a 48 6f 73 74 3a 20 77 77 77 2e 67 6f 6f 67 ..Host: www.goog
0060 6c 65 2e 6e 6c 0d 0a 55 73 65 72 2d 41 67 6e 7e le.nl..User-Agent
0070 74 3a 20 4d 6f 7a 69 6c 61 2f 35 2e 30 20 28 t: Mozilla/5.0 (

(000) ldh [12]
(001) jeq #0x800  jt 2 jf 16
...
(016) ret #0

#SharkFest16 • Computer History Museum • June 13-16, 2016
Check for source address

0000 00 12 1e bb d1 3b f8 1e df d8 87 48 08 00 45 00   ......;......H..E.
0010 02 bd 28 fb 40 00 40 06 fd 9f c0 a8 01 2b 01 01   ..(.@.@......+Bf
0020 01 01 c3 f6 00 50 f1 b8 8d 85 db 07 fd 9e 80 10   .g...P.........
0030 ff ff ce b2 00 00 01 01 08 0a 2e b9 c5 24 03 63   ...............$.c
0040 c5 41 47 45 54 20 2f 20 48 54 50 2f 31 2e 31   .AGET / HTTP/1.1
0050 0d 0a 48 6f 74 3a 20 77 77 77 2e 67 6f 6f 67   ..Host: www.goog
0060 6c 65 2e 6e 6c 0d 0a 55 73 65 72 2d 41 67 6e   le.nl..User-Agent
0070 74 3a 20 4d 6f 7a 69 6c 61 2f 35 2e 30 20 28   t: Mozilla/5.0 (

(002) ld      [26]
(003) jeqv   #0x1010101    jt 6 jf 4
Check for destination address

0000 00 12 1e bb d1 3b f8 1e df d8 87 48 08 00 45 00 .....;.....H..E.
0010 02 bd 28 fb 40 00 40 06 fd 9f c0 a8 01 2b 01 01 ..(.@.@......+Bf
0020 01 01 c3 f6 00 50 f1 b8 8d 85 db 07 fd 9e 80 18 .g...P........
0030 02 bd 28 fb 40 00 40 06 fd 9f c0 a8 01 2b 01 01 ..(.@.@......+Bf
0040 01 01 c3 f6 00 50 f1 b8 8d 85 db 07 fd 9e 80 18 .g...P........
0050 02 bd 28 fb 40 00 40 06 fd 9f c0 a8 01 2b 01 01 ..(.@.@......+Bf
0060 01 01 c3 f6 00 50 f1 b8 8d 85 db 07 fd 9e 80 18 .g...P........
0070 02 bd 28 fb 40 00 40 06 fd 9f c0 a8 01 2b 01 01 ..(.@.@......+Bf

(004) ld [30]
(005) jeq #0x1010101 jt 6 jf 16
...
(016) ret #0
Check for tcp

```
0000  00 12 1e bb d1 3b f8 1e df d8 87 48 08 00 45 00   .....;.....H..E.
0010  02 bd 28 fb 40 00 40 06 fd 9f c0 a8 01 2b 01 01   ..(.@.@......+Bf
0020  01 01 c3 f6 00 50 f1 b8 8d 85 db 07 fd 9e 80 18   .g...P...........
0030  ff ff ce b2 00 01 01 08 0a 2e b9 c5 24 03 63     .............$.c
0040  c5 41 47 45 54 20 2f 20 48 54 50 2f 31 2e 31   .AGET / HTTP/1.1
0050  0d 0a 48 6f 74 3a 20 77 77 77 2e 67 6f 6f 67   ..Host: www.goog
0060  6c 65 2e 6e 6c 0d 0a 55 73 65 4d 6f 7a 69 6c 61   le.nl..User-Agent
0070  2f 35 2e 30 20 28 00 6)      [23]
(006)  ldb      
(007)  jeq      #0x6
      jt 8 jf 16
      ...
(016)  ret      #0
```

#SharkFest16 • Computer History Museum • June 13-16, 2016
Check for ip fragmenting

```
0000  00 12 1e bb d1 3b f8 1e df d8 87 48 08 00 45 00   .....;.....H..E.
0010  02 bd 28 fb 40 00 40 06 fd 9f c0 a8 01 2b 01 01   ..(.@.@......+Bf
0020  01 01 c3 f6 00 50 f1 b8 8d 85 db 07 fd 9e 80 18   .g...P.........
0030  ff ff ce b2 00 00 01 01 08 0a 2e b9 c5 24 03 63   ............$.c
0040  c5 41 47 45 54 20 2f 20 48 54 50 2f 31 2e 31   .AGET / HTTP/1.1
0050  0d 0a 48 6f 73 74 3a 20 77 77 77 2e 67 6f 6f 67   ..Host: www.goog
0060  6c 65 2e 6e 6c 0d 0a 55 73 65 72 2d 41 67 6e 74 3a 20 4d 6f 7a 69 6c
0070  6c 61 2f 35 2e 30 20 28 (008) ldh      [20]
(009) jset     #0x1fff          jt 16   jf 10  
...  
(016) ret       #0
```
Check for port number

0000 00 12 1e bb d1 3b f8 1e df d8 87 48 08 00 45 00 ......;......H..E.
0010 02 bd 28 fb 40 00 04 06 fd 9f c0 a8 01 2b 01 01 ..(.@......+Bf
0020 01 01 c3 f6 00 50 f1 b8 8d 85 db 07 fd 9e 80 18 .g...P.........
0030 ff ff ce b2 00 00 01 01 08 0a 2e b9 c5 24 03 63 ............$.c
0040 c5 41 47 45 54 20 2f 20 48 54 54 50 2f 31 2e 31 .AGET / HTTP/1.1
0050 0d 0a 48 6f 74 3a 20 77 77 77 2e 67 6f 6f 67 ..Host: www.goog
0060 6c 65 2e 6e 6c 0d 0a 55 73 65 72 2d 41 67 6e le.nl..User-Agent
0070 74 3a 20 4d 6f 7a 69 6c 61 2f 35 2e 30 20 28 t: Mozilla/5.0 (

(010) ldxb 4*([14]&0xf)
(011) ldh [x + 14]
(012) jeq #0x50       jt 15jf 13
(013) ldh [x + 16]
(014) jeq #0x50       jt 15jf 16
(015) ret #128
(016) ret #0
offsets in BPF

(ip and host 2.2.2.2) or (vlan and ip and host 1.1.1.1)

(000) ldh [12]
(001) jeq #0x800 jt 2 jf 6
(002) ld [26]
(003) jeq #0x2020202 jt 13 jf 4
(004) ld [30]
(005) jeq #0x2020202 jt 13 jf 14
(006) jeq #0x8100 jt 7 jf 14
(007) ldh [16]
(008) jeq #0x800 jf 14
(009) ld [30]
(010) jeq #0x1010101 jt 13 jf 11
(011) ld [34]
(012) jeq #0x1010101 jt 13 jf 14
(013) ret #96
(014) ret #0

(vlan and ip and host 1.1.1.1) or (ip and host 2.2.2.2)

(000) ldh [12]
(001) jeq #0x8100 jt 2 jf 8
(002) ldh [16]
(003) jeq #0x800 jt 4 jf 15
(004) ld [30]
(005) jeq #0x1010101 jf 6
(006) ld [34]
(007) jeq #0x1010101 jf 10
(008) ldh [16]
(009) jeq #0x800 jf 15
(010) ld [30]
(011) jeq #0x2020202 jf 12
(012) ld [34]
(013) jeq #0x2020202 jf 15
(014) ret #96
(015) ret #0
offsets in BPF

VLAN 10 or VLAN 20

(000) ldh [12]
(001) jeq #0x8100 jt 2jf 5
(002) ldh [14]
(003) and #0xffff
(004) jeq #0xa jt 10jf 5
(005) ldh [16]
(006) jeq #0x8100 jt 7jf 11
(007) ldh [18]
(008) and #0xffff
(009) jeq #0x14 jt 10jf 11
(010) ret #96
(011) ret #0
offsets in BPF

\[
\text{vlan and (ether[14:2] \& 0x0fff=10 or ether[14:2] \& 0x0fff=11)}
\]

(000) ldh [12]
(001) jeq #0x8100                    jt 2 jf 7
(002) ldh [14]
(003) and #0xffff
(004) jeq #0xa                       jt 6 jf 5
(005) jeq #0xb                       jt 6 jf 7
(006) ret #96
(007) ret #0
TCP length > 0

```c
ip and tcp and
(ip[2:2]
- ((ip[0]&0x0f)<<2))
- ((tcp[12:1]&0xf0)>>2)
) > 0
```
Capture filter on steroids: SIP or SIP over IPIP

Port 5060 or

\[
\text{ip proto 4 and}
\begin{array}{l}
\text{(}
\text{ip[20+9]=17 or ip[20+9]=6}
\text{)}\\
\text{and (}
\text{ip[20+20+0:2]=5060}
\text{or}
\text{ip[20+20+2:2]=5060}
\text{)}
\end{array}
\]
Capture filter on steroids: SIP or SIP over IPIP
Taking it one step further

• Write BPF program in mnemonics
• Compile to BPF bytecode
• Use as a filter in capturing
• Why?
  - optimize filters (as the parser is good, but not perfect)
  - do stuff that is not possible with the filter parser
    (like save up to layer 4 for udp and tcp independent of option length)
• Currently not possible to use bytecode as filter in wireshark/tshark
• Use netsniff-ng tools (linux only)
; Check for ethertype 802.1Q or IP
ld [12]
jeq #0x8100,dot1q
jne #0x0800,pass14
ldb [23]
st M[0]
ldxb 4*([14]&0xf)
txa
add #14
jmp 14

; Parse the 802.1Q header
dot1q:
ld [16]
jne #0x0800,pass18
ldb [27]
st M[0]
ldxb 4*([18]&0xf)
txa
add #18

; Check L4
14:
ld M[0]
jeq #6,tcp
jeq #17,udp
txa
ret a

; UDP packet
udp:
txa
add #8
ret a

tcp:
ldh [x + 0]
jeq #80,pass
ldh [x + 2]
jeq #80,pass

; Non HTTP
ldb [x + 12]
and #0xf0
rsh #2
add x
ret a

pass14: ret #14
pass18: ret #18
pass: ret #65536
Summary

• BPF is a quick and powerful filter engine in the kernel
• Bytecode is pushed to BPF from userspace programs
• Very flexible filter language, learn the language!
• Can be used for post processing (much faster than display filters)
• Check compiled BPF program to verify working
• Write BPF program yourself for even more flexibility
• Can filter on anything (as long as you can calculate the offset of the data that you want to filter on)
• No 'searching' for data (as no loops are allowed)
Q&A

You have Questions
We have Answers
Thank You!

sake.blok@SYN-bit.nl