Using eBPF to inject IPv6 Extension Headers

https://github.com/iurmanj/ebpf-ipv6-exthdr-injection

Justin Iurman, Eric Vyncke, Benoit Donnet

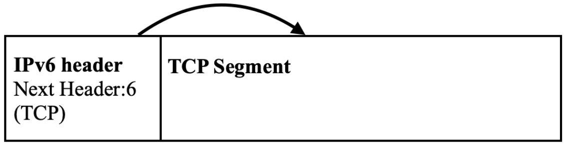
Netdev 0x17 (Vancouver)
October 31, 2023

IPv6 Extension Headers

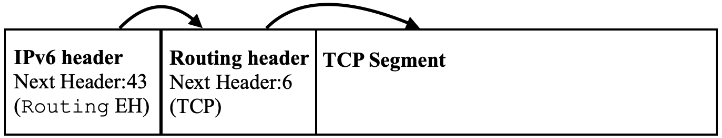
Purpose: extend the IPv6 core protocol without modification ("similar" to Options in IPv4)

<u>Examples</u>

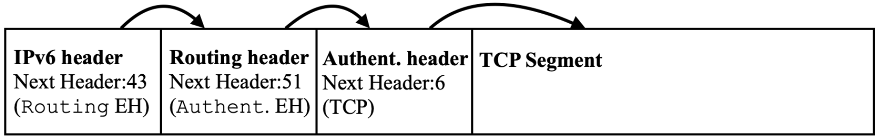
No Extension Header:



One Extension Header:



Multiple Extension Headers:



IPv6 Extension Headers (cont')

List of currently defined Extension Headers:

Protocol Number	Description	Reference
0	IPv6 Hop-by-Hop Option	[RFC8200]
43	Routing Header for IPv6	[RFC8200][RFC5095]
44	Fragment Header for IPv6	[RFC8200]
50	Encapsulating Security Payload	[RFC4303]
51	Authentication Header	[RFC4302]
60	Destination Options for IPv6	[RFC8200]
135	Mobility Header	[RFC6275]
139	Host Identity Protocol	[RFC7401]
140	Shim6 Protocol	[RFC5533]
253	Use for experimentation and testing	[RFC3692][RFC4727]
254	Use for experimentation and testing	[RFC3692][RFC4727]

Impact of Extension Headers

Two takeaway lessons*:

* Based on recent studies, including one of ours (see references in the paper)

Impact of Extension Headers

Two takeaway lessons*:

1) Size matters (yeah… sorry, it does!)

* Based on recent studies, including one of ours (see references in the paper)

Impact of Extension Headers

Two takeaway lessons*:

1) Size matters (yeah… sorry, it does!)
2) Whatever the size of a Hop-by-Hop, packets are (almost) always discarded

* Based on recent studies, including one of ours (see references in the paper)

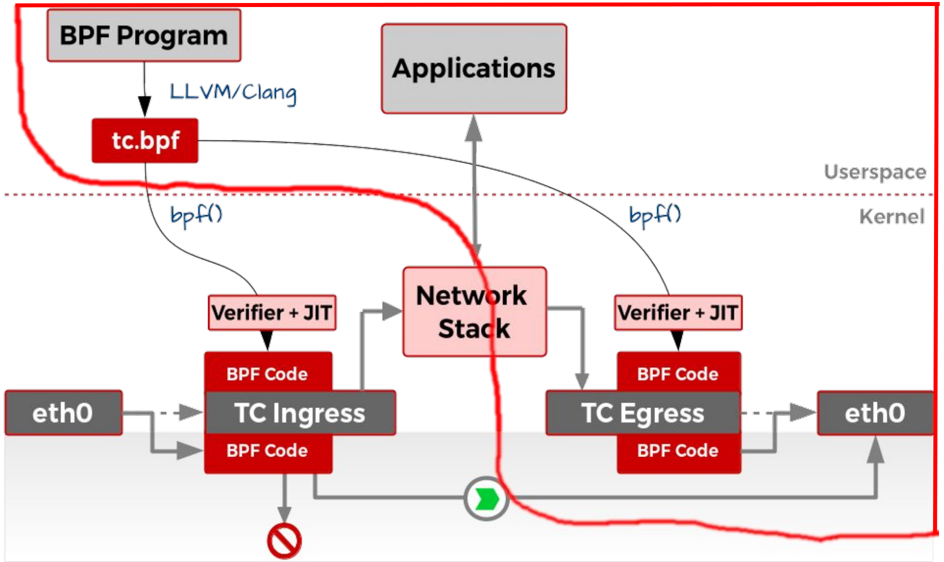
Motivations (why eBPF ?)

- We made several measurements, at the edge (IETF draft [1])
- Could be useful for others too (measurements, testing new services, etc)
- No need to modify existing tools (traceroute and similar) to inject Extension Headers
- Quick development (at least, faster than modifying each tool)
- Have a stack of Extension Headers, per packet
- No longer “just for synthetic traffic”

[1] https://datatracker.ietf.org/doc/draft-vyncke-v6ops-james

How it looks like

eBPF/tc (egress) solution



BPF (kernel) program

Sees an “opaque” data buffer to inject (same procedure regardless of the length or the content)

```
struct exthdr_t {
        struct bpf_spin_lock lock;
        __u8 ip6nexthdr;
        __u32 off_last_nexthdr;
        __u32 bytes_len;
#define MAX_BYTES 2048 /* Feel free to increase if needed */
        __u8 bytes[MAX_BYTES];
};


struct {
        __uint(type, BPF_MAP_TYPE_ARRAY);
        __uint(max_entries, 1);
        __type(key, __u32);
        __type(value, struct exthdr_t);
        __uint(pinning, LIBBPF_PIN_BY_NAME);
} MAP_NAME SEC(".maps");
```

User program

Builds the data buffer to be injected

```
Usage: ./build/tc_ipv6_eh_user.o { --disable | --enable [ --force ] EXTHDR [ EXTHDR ... EXTHDR ] }

EXTHDR := { --hbh 8..2048 | --dest 8..2048 | --rh0 24..2040 | --rh2 | --rh3 24..2040 | --rh4 24..2040
 | --fragA | --fragNA | --ah 16..1024 | --esp 16..2048 }

If a size is required, it MUST be an 8-octet multiple.
Routing Header sizes minus 8 MUST be 16-octet multiples.

Accepted chaining order, as per RFC8200 sec4.1:
 - Hop-by-Hop Options header
 - Destination Options header
 - Routing header
 - Fragment header
 - Authentication header
 - Encapsulating Security Payload header
 - Destination Options header
```

User program (cont')

- **Hop-by-Hop/Destination Options Header**:
  - "*n*" experimental options (*0x1e*) depending on the total size (*n*max_opt_size* + *remainder*)
  - random bytes for options data

- **Routing Headers** (types 0, 2, 3, 4):
  - random prefixes in range 2a00:0000::/12 (RIPE NCC)

- **Fragment Header** (both atomic, non-atomic):
  - random identification number

- **Authentication/ESP Header**:
  - random SPI, sequence number, and ICV

Example

1) Attach the program to an interface on egress with tc:
    a) *tc qdisc add dev eth0 clsact*
    b) *tc filter add dev eth0 egress bpf da obj tc_ipv6_eh_kern.o sec egress*

2) Start injecting Extension Headers (e.g., 16-byte Hop-by-Hop, 8-byte Destination, 72-byte Segment Routing):
    a) *./tc_ipv6_eh_user.o –enable –hbh 16 –dest 8 –rh4 72*

Example

```
No.     Time          Source          Destination
   2 8.245402         db02::1         2a00:ec7e:866f:5c

▾ Internet Protocol Version 6, Src: db02::1, Dst: db02::2
    0110 .... = Version: 6
  ▸ .... 0000 0000 .... .... .... .... .... = Traffic Class: 0x00 (
    .... 1110 0010 1110 0111 1100 = Flow Label: 0xe2e7c
    Payload Length: 160
    Next Header: IPv6 Hop-by-Hop Option (0)
    Hop Limit: 64
    Source Address: db02::1
    Destination Address: db02::2
  ▾ IPv6 Hop-by-Hop Option
      Next Header: Destination Options for IPv6 (60)
      Length: 1
      [Length: 16 bytes]
    ▸ Experimental (0x1E)
  ▾ Destination Options for IPv6
      Next Header: Routing Header for IPv6 (43)
      Length: 0
      [Length: 8 bytes]
    ▸ Experimental (0x1E)
  ▾ Routing Header for IPv6 (Segment Routing)
      Next Header: ICMPv6 (58)
      Length: 8
      [Length: 72 bytes]
      Type: Segment Routing (4)
      Segments Left: 4
      Last Entry: 3
      Flags: 0x00
      Tag: 0000
      Address[0]: 2a00:ec7e:866f:5c08:ebcd:bec:f5a3:4110
      Address[1]: 2a00:fcef:5326:8cf7:90d3:8f85:b75a:a587
      Address[2]: 2a00:1c85:5c08:3e2:785f:ea63:2cf6:4f22
      Address[3]: 2a00:9990:3296:8086:bc0c:7d4c:e00d:d197
```

Next

Currently:

1. a custom filter is built into the ebpf program and must be modified as needed
2. randomly generated data
3. fake AH/ESP (no real encryption)

Potential solution:

1. combine the “*tc filter add […] egress bpf da obj [...]*” with a tc filter on proto/ports (possible?)
2. pass “real” data as a config file (e.g., based on a yang model or similar) to the user program
3. add specific “post-processing” per packet

Thank you

https://github.com/iurmanj/ebpf-ipv6-exthdr-injection