IPv6 Extension Headers: Devil or Angel in Disguise?

Eric Vyncke, Distinguished System Engineer, evyncke@cisco.com

@evyncke

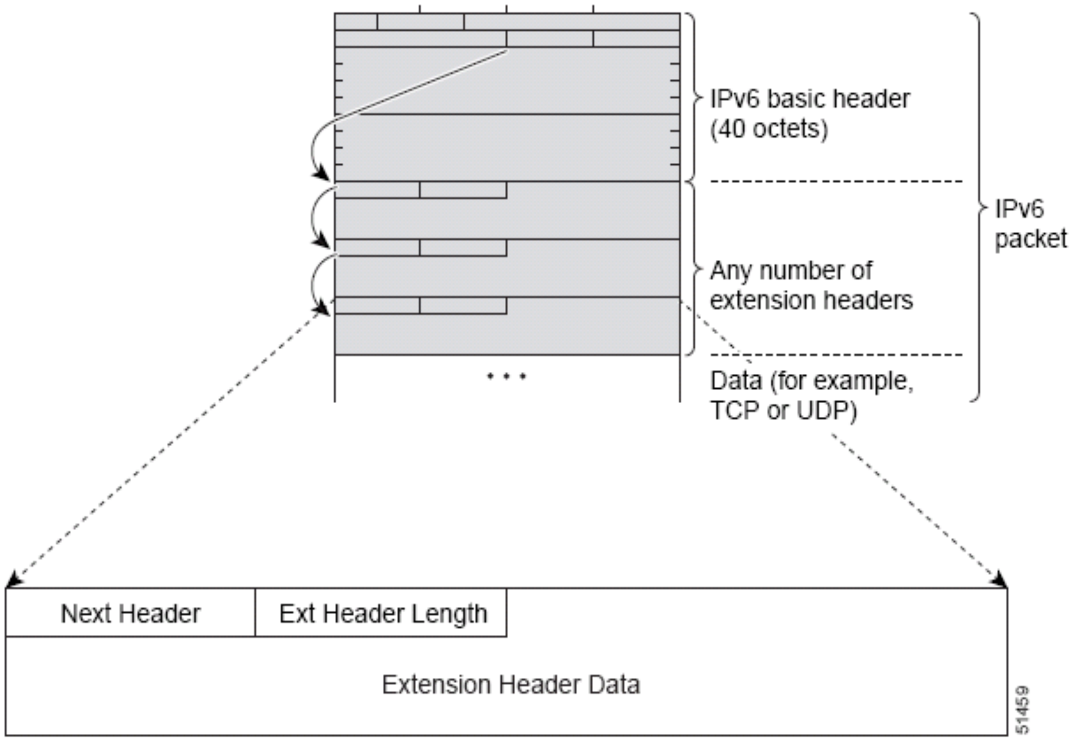
June 2016

Extension Header Lab with Scapy

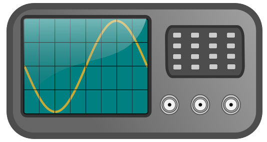
IPv6 Extension Header

Some protocols that require Ext Headers:

- IPv6 Fragmentation
- IPsec (AH and ESP)
- Mobile IPv6
- RPL (RFC 6554)
- Segment Routing
- iOAM6

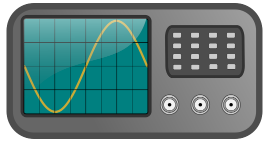


Packet Forgery with SCAPY



- Scapy is a open source packet forgery tool built on Python
- Powerful albeit complex to understand and to use:

```
evyncke@host1:~# scapy
Welcome to Scapy (2.1.0)
>>> target="2001:db8:23:0:60de:29ff:fe15:2”
>>> packet=IPv6(dst=target)/ICMPv6EchoRequest(id=0x1234, seq=RandShort(),
  data="ERIC")
>>> sr1(packet)
Begin emission:
Finished to send 1 packets.
Received 2 packets, got 1 answers, remaining 0 packets
<IPv6  version=6L tc=0L fl=0L plen=12 nh=ICMPv6 hlim=62
  src=2001:db8:23:0:60de:29ff:fe15:2 dst=2001:db8:1:0:60de:29ff:fe15:1 |
  <ICMPv6EchoReply  type=Echo Reply code=0 cksum=0xdb04 id=0x1234 seq=0x956a
  data='ERIC' |>>
```



Let’s Try it With Routing Header 0 & Tcpdump

```
a="2001:DB8:1::1"
b="2001:DB8:23::2”
route=[]
for i in range(0, 30):
  route.append(a)
  route.append(b)
packet=IPv6(dst=b,hlim=255)/IPv6ExtHdrRouting(addresses=route,type=0)/ICMPv6EchoRequest()
sr1(packet)
```

Using a recent IOS, the router refuses to process Routing Header Type 0

```
IP6 (hlim 63, next-header ICMPv6 (58) payload length: 384) 2001:db8:23::2 > scapy_host: [icmp6
  sum ok] ICMP6, parameter problem, length 384, errorneous - octet 42
```

Fragmentation Used in IPv4 by Attackers ... Also applicable to IPv6 of course

- Great evasion techniques
  - Some firewalls do not process fragments except for the first one
  - Some firewalls cannot detect overlapping fragments with different content
- IPv4 tools like whisker, fragrout, etc.
- Makes firewall and network intrusion detection harder
- Used mostly in DoSing hosts, but can be used for attacks that compromise the host
  - Send a fragment to force states (buffers, timers) in OS

  - See also: http://insecure.org/stf/secnet_ids/secnet_ids.html 1998!

Parsing the Extension Header Chain Fragments and Stateless Filters

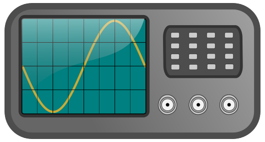
- RFC 3128 is not applicable to IPv6
- Layer 4 information could be in 2^{nd} fragment
- But, stateless firewalls could not find it if a previous extension header is fragmented

IPv6 hdr	HopByHop	Routing	Fragment1	Destination …		
IPv6 hdr	HopByHop	Routing	Fragment2	… Destination	TCP	Data

Layer 4 header is in 2^{nd} fragment, Stateless filters have no clue where to find it!

- But, RFC6980: "nodes MUST silently ignore NDP … if packets include a fragmentation header
- But, RFC7112: "A host that receives a First Fragment that does not satisfy… SHOULD discard the packet

Fragment Obfuscation with Scapy & tcpdump

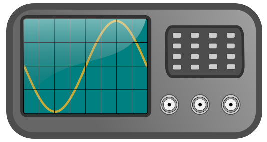


```
>>> packet=IPv6(dst=dst)/IPv6ExtHdrDestOpt(options=PadN(optdata='A'*20))/
  TCP(sport=sport,dport=22,flags="S", seq=100)
>>> frag1=IPv6(dst=dst)/IPv6ExtHdrFragment(nh=60, id=0xabbababe, m=1, offset=0)/str(packet)
  [40:48]
>>> frag2=IPv6(dst=dst)/IPv6ExtHdrFragment(nh=60, id=0xabbababe, m=0, offset=1)/str(packet)
  [48:84]
>>> send(frag1)
>>> send(frag2)
```

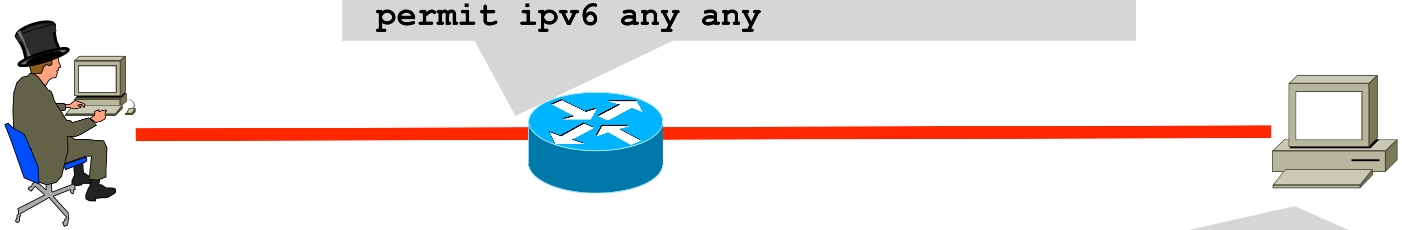
```
IP6 (hlim 64, next-header Fragment (44) payload length: 16) 2001:...:1 > 2001:...:2: frag (0xabbababe:0|8) [|DSTOPT]
   0x0000:  6000 0000 0010 2c40 2001 0db8 0001 0000  `.....,@........
   0x0010:  60de 29ff fe15 0001 2001 0db8 0023 0000  `.)..........#..
   0x0020:  60de 29ff fe15 0002 3c00 0001 abba babe  `.).....<.......
   0x0030:  0602 0114 4141 4141                      ....AAAA

IP6 (hlim 64, next-header Fragment (44) payload length: 44) 2001:...:1 > 2001:...:2: frag (0xabbababe:8|36)
   0x0000:  6000 0000 002c 2c40 2001 0db8 0001 0000  `....,,@........
   0x0010:  60de 29ff fe15 0001 2001 0db8 0023 0000  `.)..........#..
   0x0020:  60de 29ff fe15 0002 3c00 0008 abba babe  `.).....<.......
   0x0030:  4141 4141 4141 4141 4141 4141 4141 4141  AAAAAAAAAAAAAAAA
   0x0040:  47b3 0016 0000 0064 0000 0000 5002 2000  G......d....P...
   0x0050:  da35 0000
```

Let’s Try the Naïve ACL...



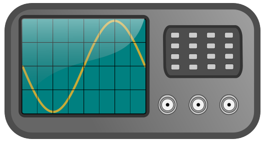
```
ipv6 access-list NO_SSH
 deny tcp any any eq 22 log
 permit ipv6 any any
```



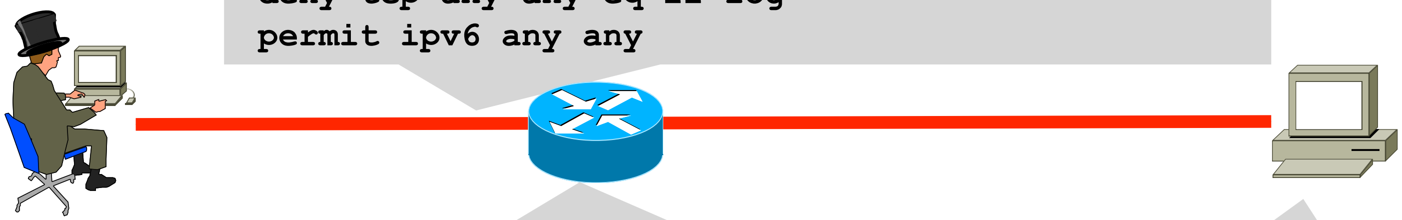
```
IP6 (hlim 62, next-header Fragment (44) payload length: 16) 2001:..:1 > 2001:..:2: frag
(0xabbababe:0|8) [|DSTOPT]
IP6 (hlim 62, next-header Fragment (44) payload length: 44) 2001:..:1 > 2001:..:2: frag
(0xabbababe:8|36)

SSH accepts connection and replies
IP6 (hlim 64, next-header TCP (6) payload length: 24) 2001:...:2.22 > 2001:...:1.18355: Flags
[S.], cksum 0x138c (correct), seq 621319016, ack 101, win 5760, options [mss 1440], length 0
```

Let’s Try undetermined_transport...



```
ipv6 access-list NO_SSH2
 deny ipv6 any any undetermined-transport log
 deny tcp any any eq 22 log
 permit ipv6 any any
```



```
%IPV6_ACL-6-ACCESSLOGSP: list NO_SSH2/10 denied tcp
2001:...:1 -> 2001:...:2, 1 packet
```

1st fragment is not received..

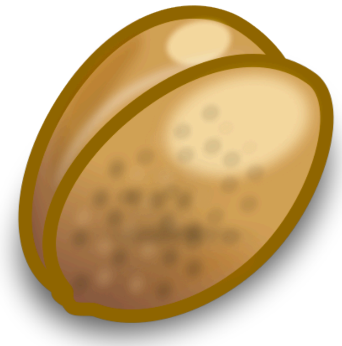
```
IP6 (hlim 62, next-header Fragment (44) payload length: 44) 2001:..:1 > 2001:..:2: frag
(0xabbababe:8|36)
```

Reassembly fails after time-out, connection is never established

Extension Headers for Segment Routing

Segment Routing in a Nutshell

- Segment Routing:
  - **Source based routing model** where the source chooses a path and encodes it in the packet header as an ordered list of segments
  - A segment is effectively an instruction applied to the packet as it traverses its list of segments
  - Segment Routing leverages the source routing architecture defined in RFC2460 for IPv6, including the use of the IPv6 Routing Extension Header



Source: wikimedia

Segment Routing and the Source Based Routing Model

- Segment Routing technology is extensively explained in
  - http://www.segment-routing.net (includes all published IETF drafts)
- Segment Routing data-planes
  - SR-MPLS: segment routing applied to MPLS data-plane
  - **SR-IPv6: segment routing applied to IPv6**
- SR-IPv6 allows Segment Routing do be deployed over non-MPLS networks and/or in areas of the network where MPLS is not present (e.g.: datacenters)
- Segment Routing backward compatibility
  - SR nodes fully interoperate with non-SR nodes
  - No need to have a full network upgrade

Segment Routing Header

- Segment Routing introduces a new Routing Header Type:
  - The Segment Routing Header (SRH)
  - Contains the list of segments the packet should traverse
  - VERY close to what already specified in RFC2460
  - Changes are introduced for:
    > Better flexibility
    > Addressing security concerns raised by RFC5095
- Three SR-IPv6 drafts:
  - draft-ietf-6man-segment-routing-header
  - ~~draft-vyncke-6man-segment-routing-security~~
  - draft-ietf-spring-ipv6-use-cases

S. Previdi, Ed.
C. Filsfils
E. Vyncke
Cisco Systems, Inc.
Comcast
Rogers Communications
D. Lebrun
Universite Catholique de Louvain
March 18, 2016

IPv6 Segment Routing Header (SRH)
draft-ietf-6man-segment-routing-header-01

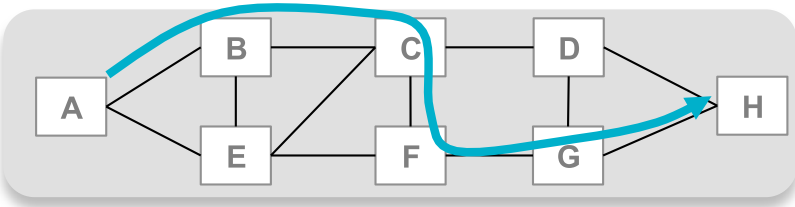
J. Brzozowski
J. Leddy
Comcast
I. Leung
Rogers Communications
S. Previdi
M. Townsley
C. Martin
C. Filsfils
D. R. Maglione, Ed.
Cisco Systems
March 3, 2016

Source Packet Routing in Networking

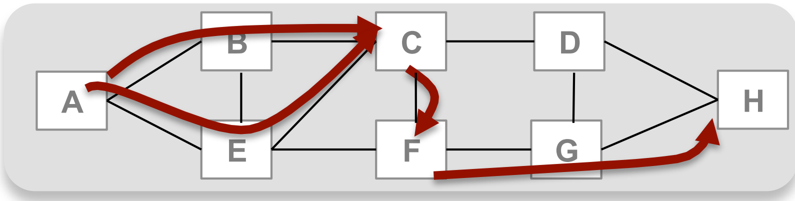
IPv6 SPRING Use Cases
draft-ietf-spring-ipv6-use-cases-06

Segment Routing Model

- Assuming following topology:
  – Node A has two shortest paths to C



- How to best express path: [A, B, C, F, G, H]
- Source routed path with segments: [C,F,H]
  > First segment: set of shortest paths from A to C (ECMP aware)
  > Second segment: adjacency/link from C to F
  > Third segment: shortest path from F to H



SRH: identical to RFC 2460

- Next Header: 8-bit selector. Identifies the type of header immediately following the SRH
- Hdr Ext Len: 8-bit unsigned integer. Defines the length of the SRH header in 8-octet units, not including the first 8 octets
- Routing Type: TBD by IANA (SRH)
- Segment Left: index, in the Segment List, of the current active segment in the SRH. Decremented at each segment endpoint.

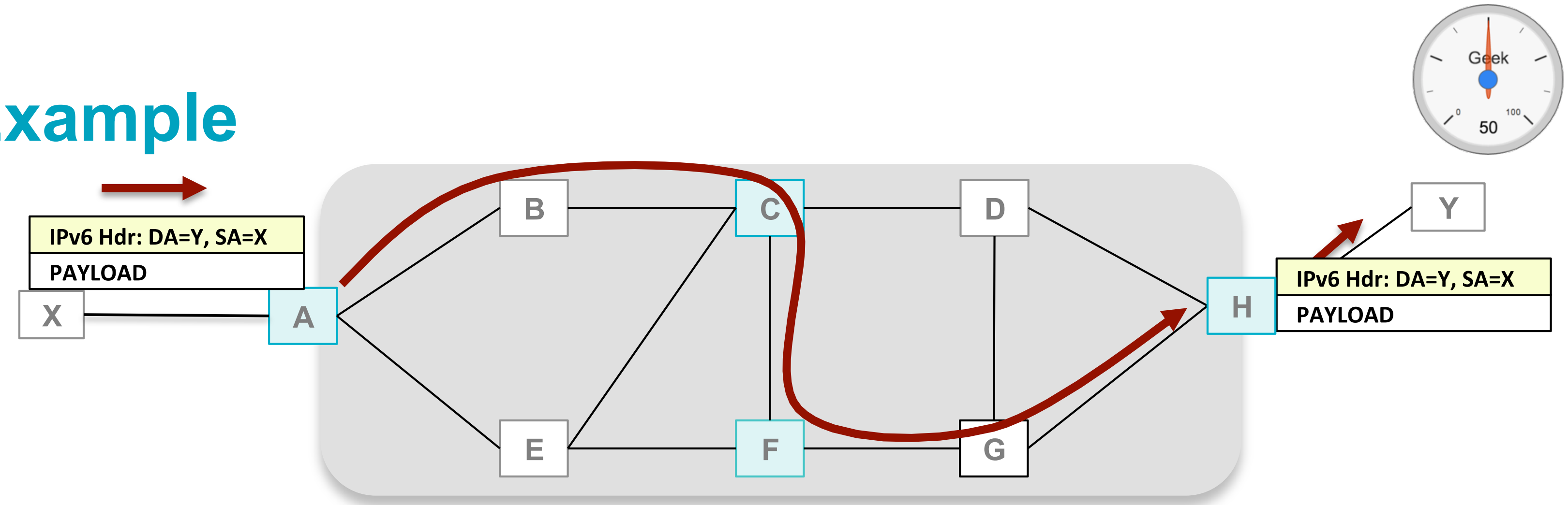
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Header   |  Hdr Ext Len  | Routing Type  | Segments Left |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| First Segment |             Flags             |   RESERVED    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|            Segment List[0] (128 bits IPv6 address)            |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
                              ...
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|            Segment List[n] (128 bits IPv6 address)            |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//                                                             //
//         Optional Type Length Value objects (variable)       //
//                                                             //
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

SRH: New

- First Segment: offset in the SRH, not including the first 8 octets and expressed in 16-octet units, pointing to the last element of the Segment List
- Flags: HMAC key present, OAM (see later), Clean (remove SRH at egress), ...
- Segment List[n]: 128 bit IPv6 addresses representing each segment of the path. The segment list is encoded in the reverse order of the path: the last segment is in the first position of the list and the first segment is in the last position
- TLV objects (optional): to mark ingress/ ingress SR address, to remember original source address, HMAC key (for security)

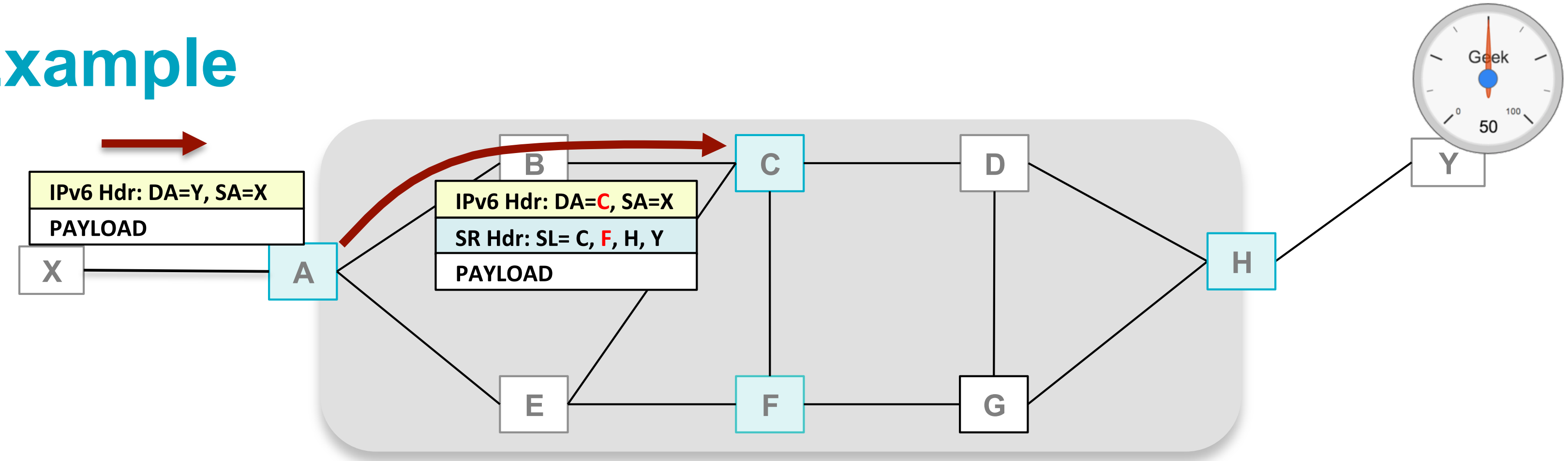
```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Next Header   |  Hdr Ext Len  | Routing Type  | Segments Left |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| First Segment |           Flags               |  RESERVED     |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|            Segment List[0] (128 bits IPv6 address)            |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|                                                               |
                              ...
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                                                               |
|            Segment List[n] (128 bits IPv6 address)            |
|                                                               |
|                                                               |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
//                                                             //
//         Optional Type Length Value objects (variable)       //
//                                                             //
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

SR-IPv6 Example



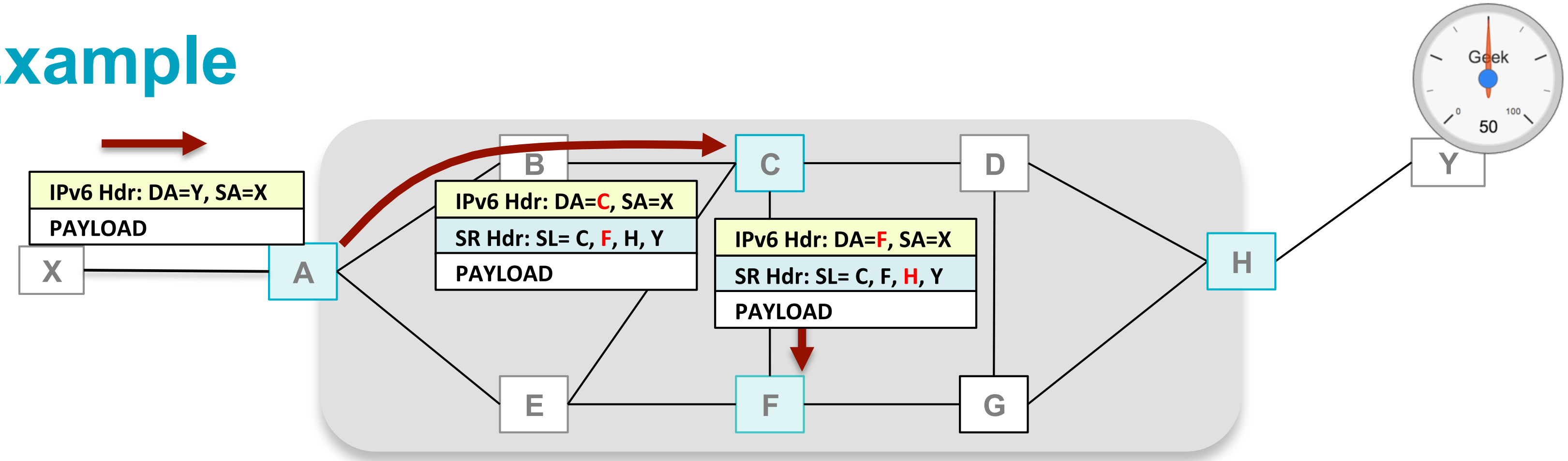
- Example:
  - Classify packets coming from X and destined to Y and forward them across A,B,C,F,G,H path
  - Nodes A, C, F and H are SR capable

SR-IPv6 Example



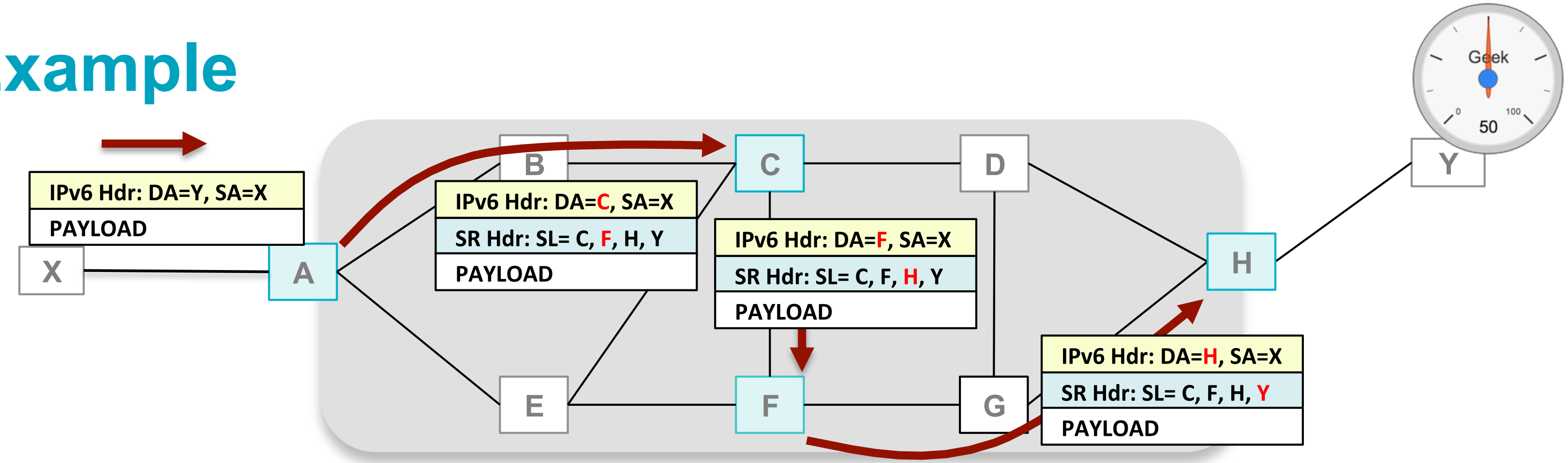
- At ingress, the Segment Routing Header (SRH) contains
  - **Segment List**: C,F,H,Y (original destination address is encoded as last segment of the path)
  - **Segments Left**: identities the next segment of the path (F)
  - **DA** is set as the address of the first segment: C
- Packet is sent towards its DA (C, representing the first segment)
  - Packet can travel across non SR nodes who will just ignore the SRH
  - **RFC2460 mandates only the node in the DA must examine the SRH**

SR-IPv6 Example



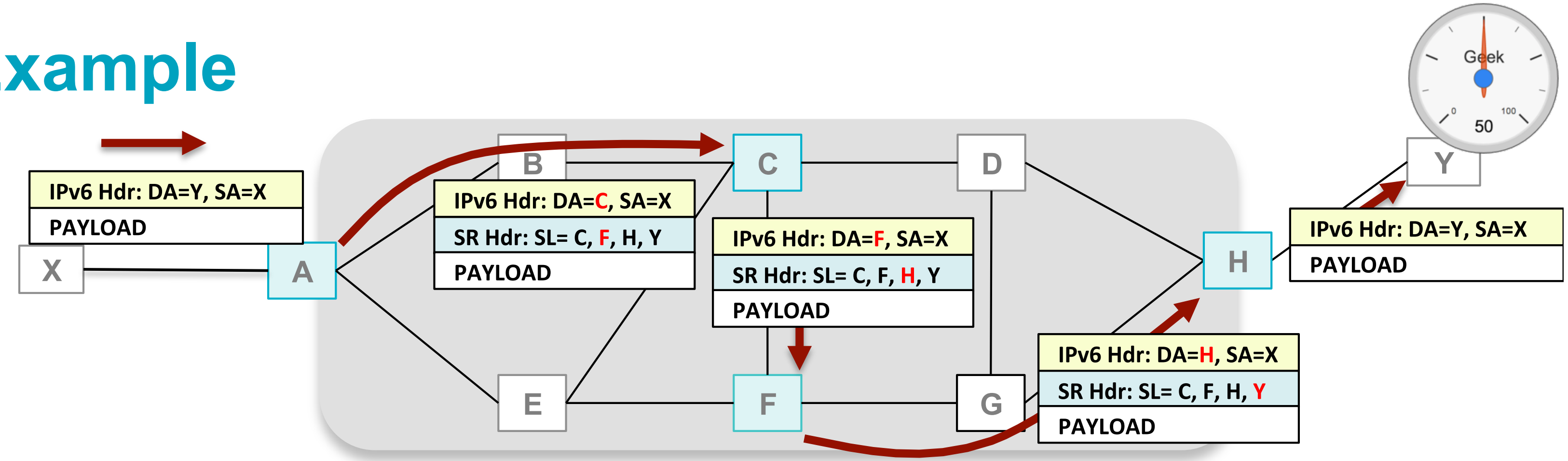
- When packet reaches the segment endpoint C
  - Segment Left is inspected and used in order to update the DA with the next segment address: F
  - Segment Left is decremented: now indicates next segment: H
  - Packet is sent towards its DA

SR-IPv6 Example



- When packet reaches the segment endpoint F the same process is executed:
  - Segment Left is inspected and used in order to update the DA with the next segment address: H
  - Segment Left is decremented: indicated next as Y (the original DA)
  - Packet is sent towards its DA

SR-IPv6 Example

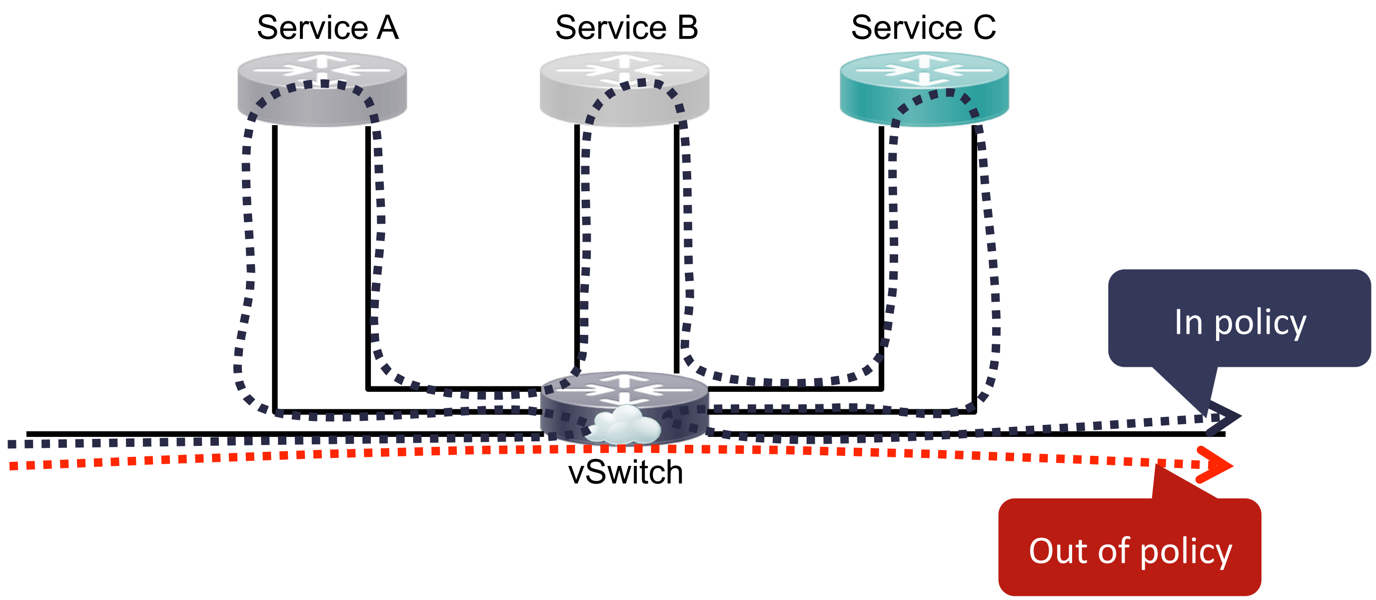


- When packet reaches the segment endpoint H:
  - Segment Left is inspected (== 0) and used in order to update the DA with the next segment address: Y
  - An optional flag (cleanup-flag) in SRH tells H to cleanup the packet and remove the SRH
  - Packet is sent towards its DA

Extension Headers for iOAM

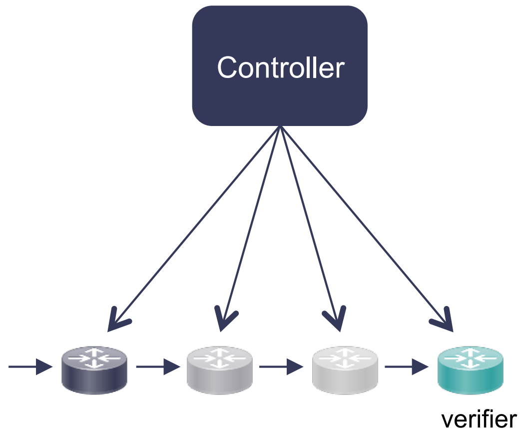
Ensuring Service Chain and Path Integrity

Service Chain: A➧B➧C

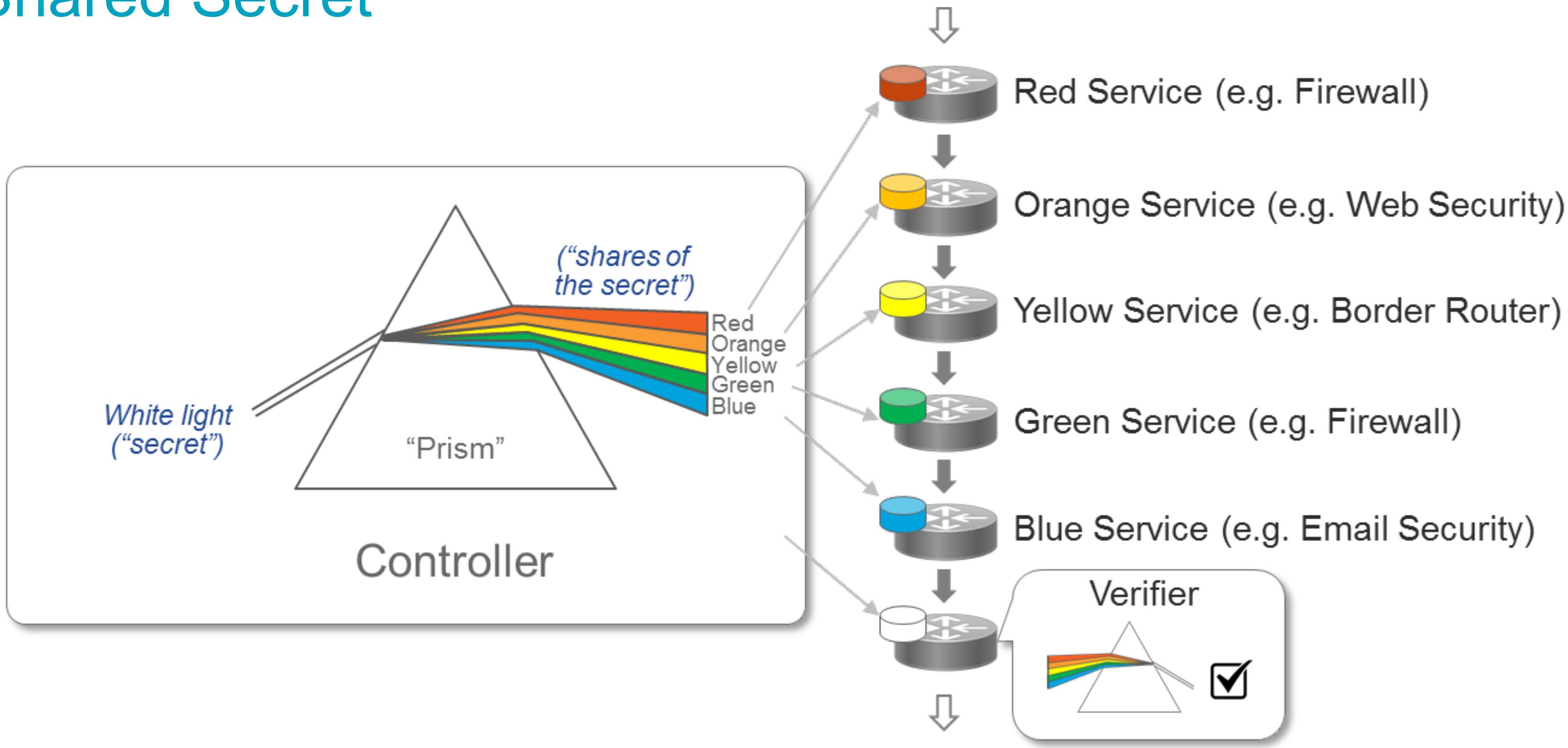


Service Chain Integrity Validation: Approach

- Add meta-data to all packets that traverse a service chain
- The added meta-data allows a verifying node (egress node) to check whether a packet traversed the service chain correctly or not
- Security mechanisms are used on the meta-data to protect against incorrect or misuse (i.e. configuration mistakes, people playing tricks with routing, capturing, spoofing and replaying packets).
- The meta-data is secured through the use of keys. Service functions retrieve the keys from a controller over a secure channel.

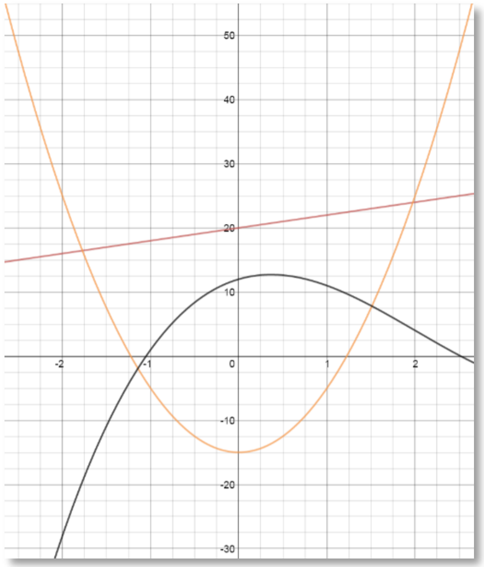


Service Chain Integrity Validation Concept
Shared Secret



Solution Approach: Leveraging Shamir's Secret Sharing

Polynomials 101



$$f2(x) = 10x^2 - 15$$

- Parabola: Min 3 points

$$f1(x) = 2x + 20$$

- Line: Min 2 points

$$f3(x) = x^3 - 6x^2 + 4x - 12$$

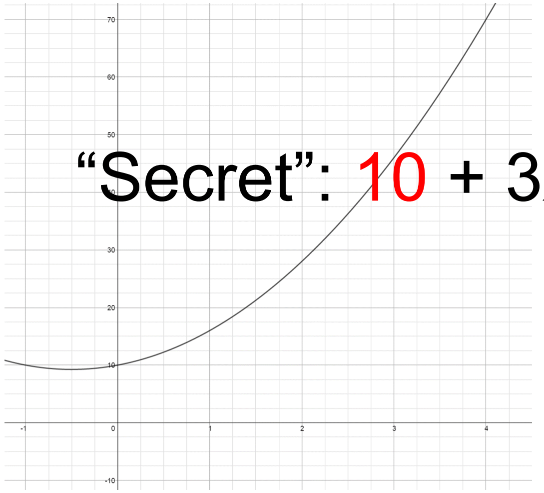
- Cubic function: Min 4 points

General: It takes k+1 points to defines a polynomial of degree k.

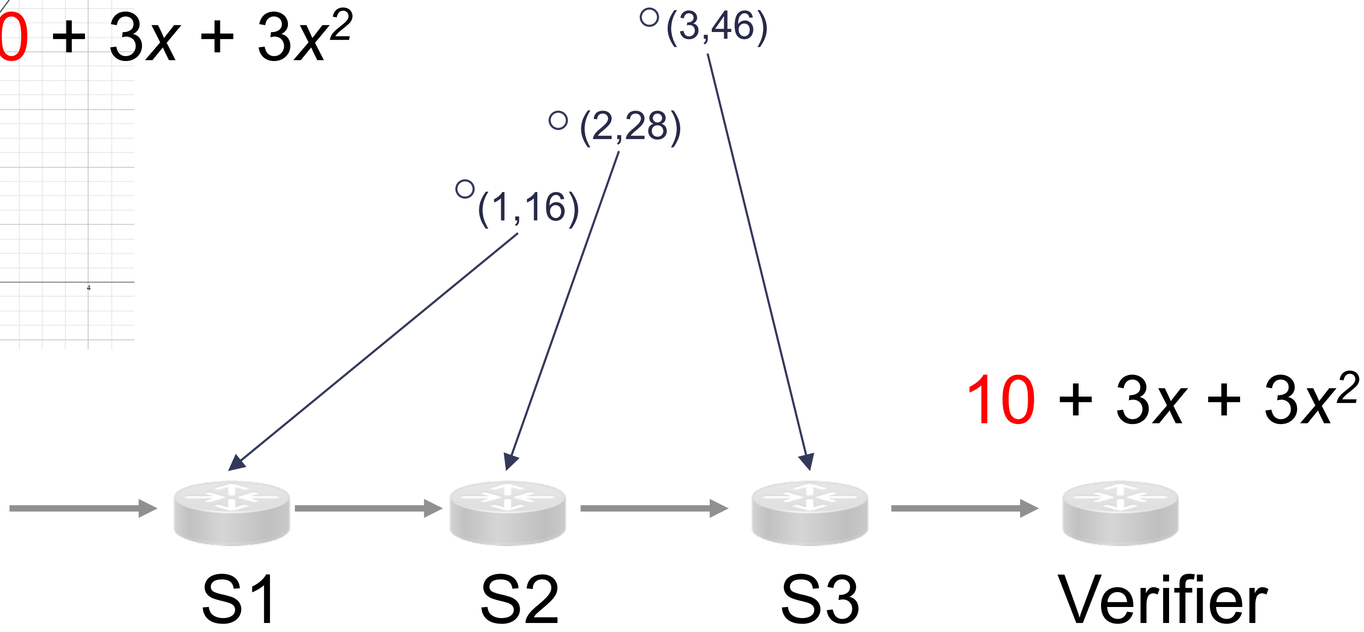
Credits: fbrockne@cisco.com

Solution Approach: Leveraging Shamir's Secret Sharing

Idea Concept



"Secret": $\color{red}{10} + 3x + 3x^2$



Credits: fbrockne@cisco.com

Solution Approach: Leveraging Shamir’s Secret Sharing

- Outline :
  - Each service is given a point on the curve
  - When the packet travels through each service it collects these points
  - A verifier can reconstruct the curve using the collected points
  - If there are k+1 services and k+1 points chosen, then the verifier can construct
    k degree polynomial and verify.
  - The polynomial cannot be constructed if a few points are missed. Any lesser points means few services are missed!
- Concern: Operationally complex to configure and recycle so many curves and their respective points for each service function

Credits: fbrockne@cisco.com

Simpler & Faster with 2 Polynomials

- POLY-1 secret, constant per chain:
  - $a_1 + b_1 x + c_1 x^2 + \ldots$ (only known by verifier)
  - Each service gets a point on POLY-1 (for x = 1, 2, ...)
- POLY-2 public, with RND-2 random and per packet
  - *RND-2* $+ b_2 x + c_2 x^2 + \ldots$ (known by all services + verifier)
  - Each service generates a point on POLY-2 each time a packet crosses it (same *x* as in POLY-1)
- Each service adds the two points to get a point on POLY-3 and passes it to verifier by adding it to each packet.
- The verifier constructs POLY-3 from the points given by all the services and cross checks whether POLY-3 = POLY-1 + POLY-2
- *Computationally efficient: Only 3 additions and 1 multiplication per hop*
- *All operations are done in a finite field (modulo prime)*

POLY-1
Secret – Constant

+

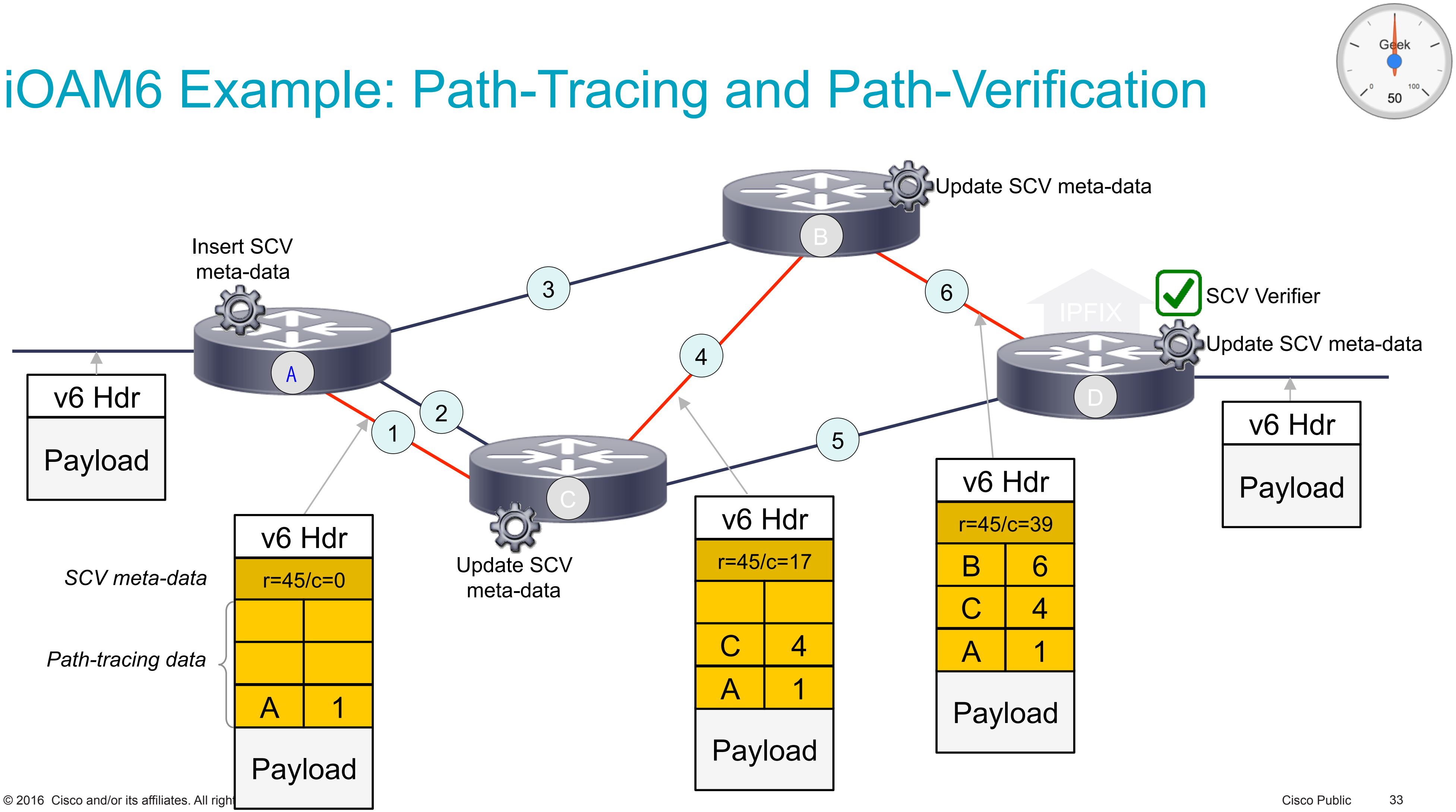
POLY-2
Public – Per Packet

=

POLY-3
Secret – Per Packet

Credits: fbrockne@cisco.com

iOAM6 Example: Path-Tracing and Path-Verification



Security Considerations

- An attacker bypassing few services, will miss adding a respective point on POLY-1 to corresponding point on POLY-2 , thus the verifier cannot construct POLY-3 for cross verification
- An attacker watching values, doing differential analysis across service functions (i.e. as the packets entering and leaving), cannot construct a point on POLY-1 as the operations are done over a finite field (i.e. modulo prime).
- Replay attacks could be avoided by carefully choosing POLY-2. It could be a timestamp concatenated with a random string.
- The proofs of correctness and security are based on Shamir's Secret Sharing Scheme .

Credits: fbrockne@cisco.com

Extension Headers Policy? Forward? Drop ?

Extension Header Security Policy

- White list approach for your traffic
  - Only allow the REQUIRED extension headers (and types), for example:
    - Fragmentation header
    - Routing header type 2 & destination option (when using mobile IPv6)
    - IPsec ☺ AH and ESP
    - And layer 4: ICMPv6, UDP, TCP, GRE, ...
  - If your firewall is capable:
    - Drop 1st fragment without layer-4 header
    - Drop routing header type 0
    - Drop/ignore hop-by-hop

Source: Tony Webster, Flickr

Extension Header Loss over the Internet

- End users SHOULD filter packets with extension headers
- But, what are your ISP and its transit provider doing to your packets?

Source: Paul Townsend, Flickr

- draft-gont-v6ops-ipv6-ehs-in-real-world
  - About 20-40% of packets with Ext Hdr are dropped over the Internet

Previous Extension Headers Research by Others

- IETF-88, Nov-2013, fgont-iepg-ietf88-ipv6-frag-and-eh.pdf
  - *“Fragmentation and Extension Header Support in the IPv6 Internet”*
  - Single origin, destination = Alexa top web sites (883 unique addr)
  - Ext header size: 8 bytes and 1024 bytes
  - Failure rate: 45%
- IETF-89, with Tim Chown: 60% packet drops
- IETF-90, Jul-2014, iepg-ietf90-ipv6-ehs-in-the-real-world-v2.0.pdf
  - *“IPv6 Extension Headers in the Real World v2.0”*
  - Origin: RIPE Atlas probes, destination = Alexa again
  - Ext header size: 8, 256, 512 and 1024 bytes
  - Failure rate: between 60% and 90%

Issues with Previous Experiments

- Destination: big web sites (Alexa)
- It is expected that destination drops what is unexpected
- Outdated by 9 months in early 2015
- Not testing about Routing Header (for segment routing)
- Not matching other empirical tests

Methodology of our study

1. Determine a set of IPv6 addresses to test :
   - From Alexa's Top 1 Million list
   - From IPv6 BGP-advertised prefixes
2. TCP Traceroute without EHs :
   - Send v6 packets with TCP payload to port 80 of the destination with varying TTL => Routers in the path answer with ICMPv6 Time Exceeded
3. TCP Traceroute with EHs:
   - Same thing but adding an Extension Header before the TCP payload
4. Analysing the traceroutes

Methodology of our research :
Step 1) Determining a set of IPv6 addresses to test

- From Alexa's Top 1 Million list :
  - Take those that have a AAAA record
  - … with a reachable IPv6 address in the AAAA record

- From BGP-advertised IPv6 prefixes
  - Address = [prefix]::1
  - Doesn't exist ? No problem, we are supposed to reach the AS -> Enough

Methodology of our research May 2015 : 2) TCP Traceroute with EHs

First, normal TCP traceroute without EH, then with EH

EH set :

- Destination Option Header
  16, 256, 512 bytes
- Hop-by-Hop Header
  16 bytes
- DO 16B + HbH 16B
- Routing Header type 4 (expected for Segment Routing)
- Fragment Header
  Normal and Atomic

EHs blocked by our ISP (so no result) :

- Hop-by-Hop Header
  256, 512 bytes
- Routing Header type 0 (deprecated)

Methodology of our study : Analysing the traceroutes

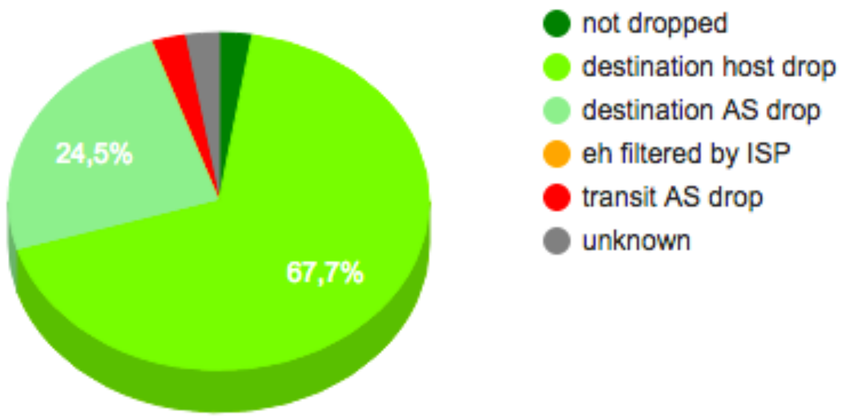
- Is it a problem ? Depends where it was dropped !
  - If dropped by the destination organization (host or same AS): Not a problem !
  - If dropped in transit: not cool…

- Where is the dropping node ?
  - If IP corresponds to some major IXPs, we look up the corresponding ASN by knowing the addressing logic, or in a database
  - Otherwise, normal GeoIP ASN lookup

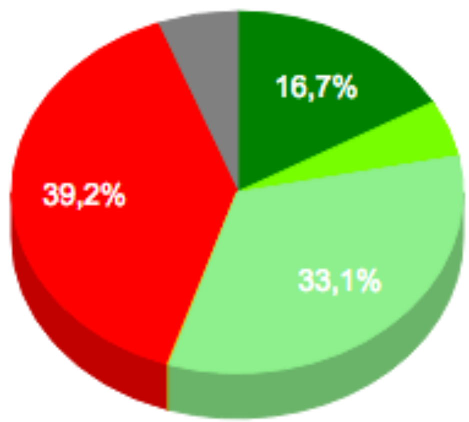
Results and analysis

- Drop rates depend on the Extension Header

D.O. 16B

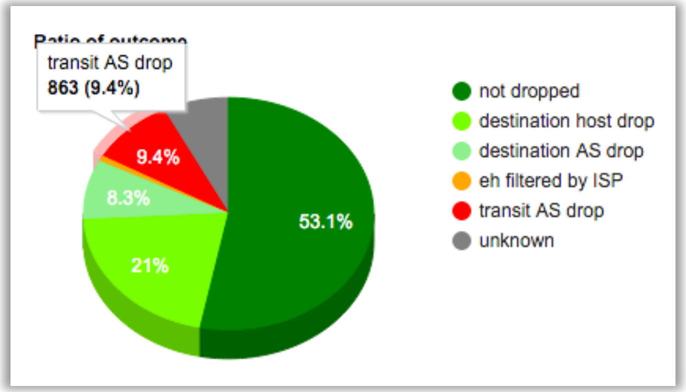


HbH 16B

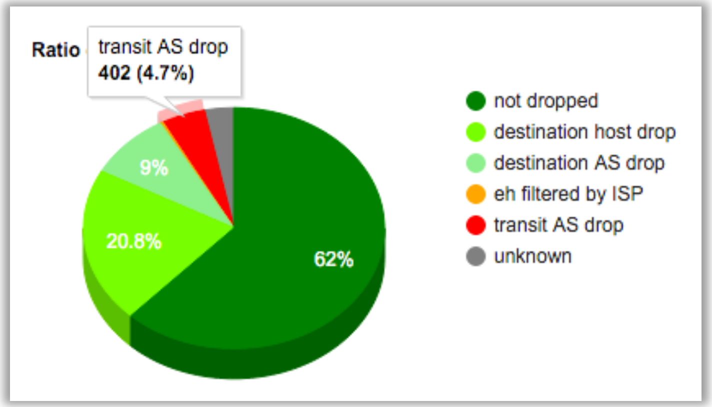


For Alexa

Things Keeps Improving Though



BGP in Spring 2015



BGP in Spring 2016

- Current research by Polytechnique Paris (Mehdi Kouhen) and Cisco (Eric Vyncke)
  - And VM provided by Sander Steffann
- https://btv6.vyncke.org/exthdr/index.php?ds=bgp2016&t=fh (work in progress!)
- http://evyncke.go6lab.si/exthdr/index.php

Summary

Summary

- Extension headers are useful to extend IPv6
  - Good old IPsec
  - New functions: segment routing, iOAM
- Let's not be naïve though
  - Do we need fragments?
  - Transit providers: do not harm extension headers
  - Internet edge: use a strict white list approach

IPv6 Business Conference

Organized by SWISS IPv6 COUNCIL

2016 June, 16

Thanks to all our Sponsors

Infoblox CONTROL YOUR NETWORK

f5

SWITCH

DIGICOMP

eb-Qual

itrain THE ESSENCE OF IT

Sunny Connection

BLUECAT

asecus securing your business

flowdays achieving more together

Thank you.