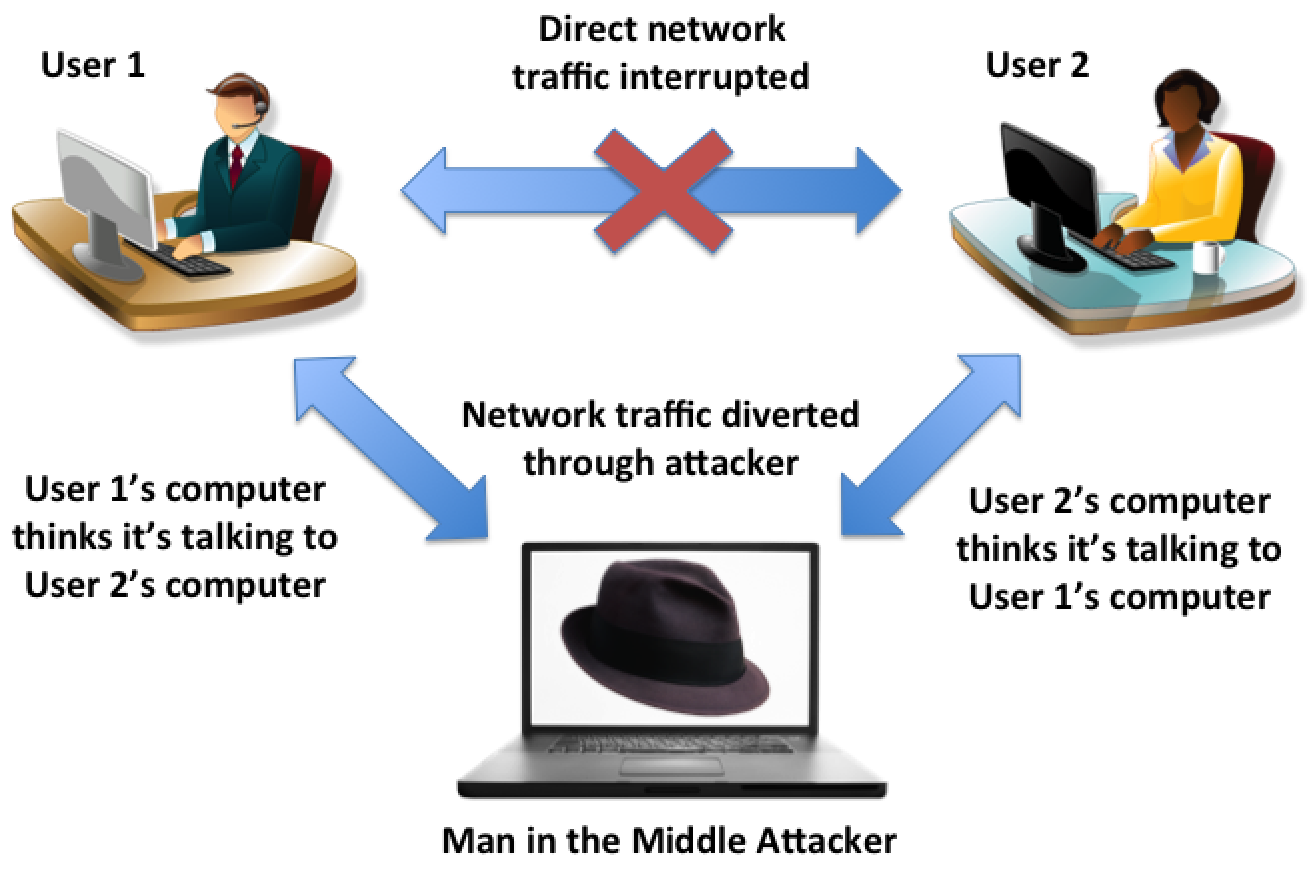
Man-in-the-Middle Attack

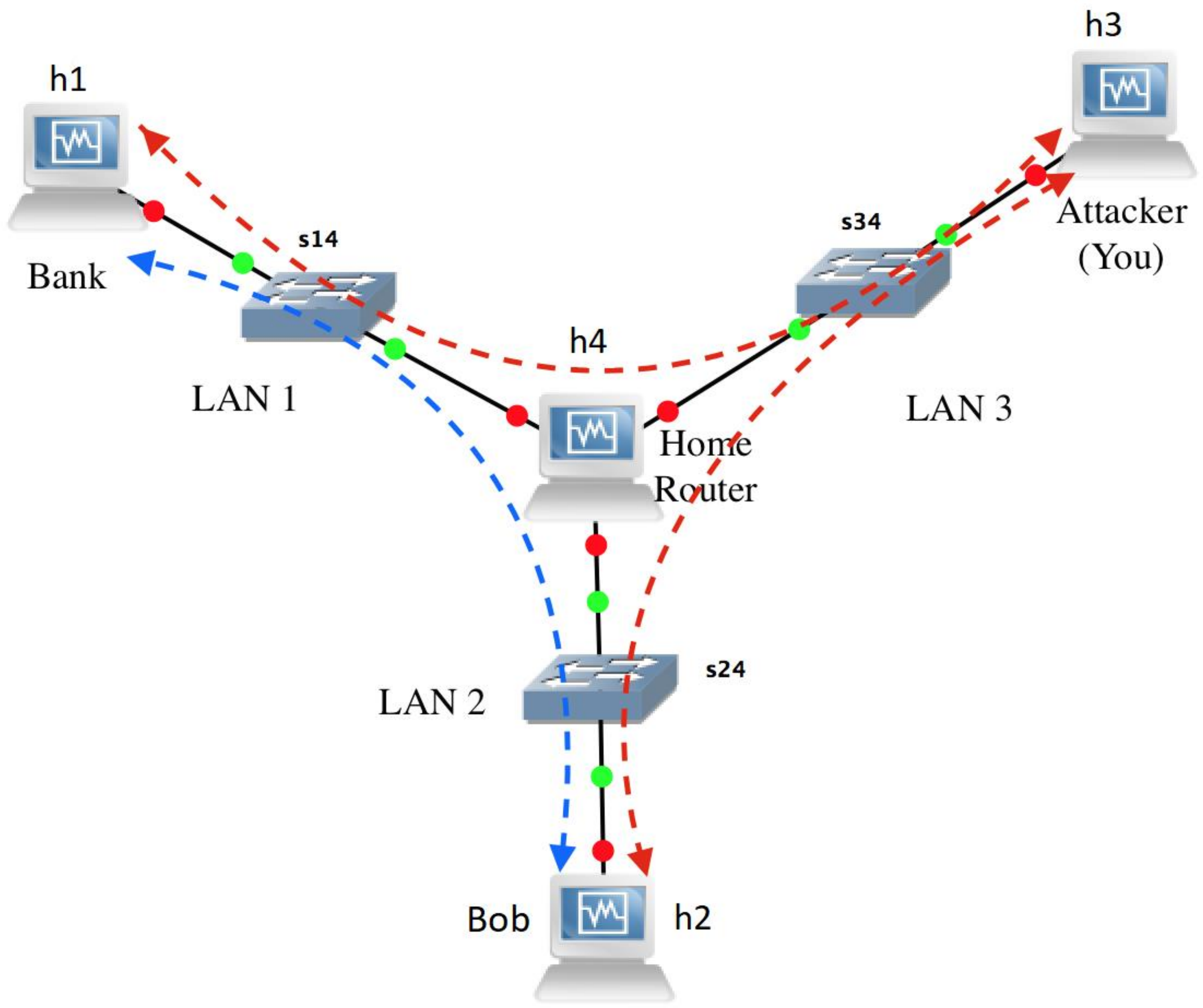
Man-in-the-middle (MITM) Attack

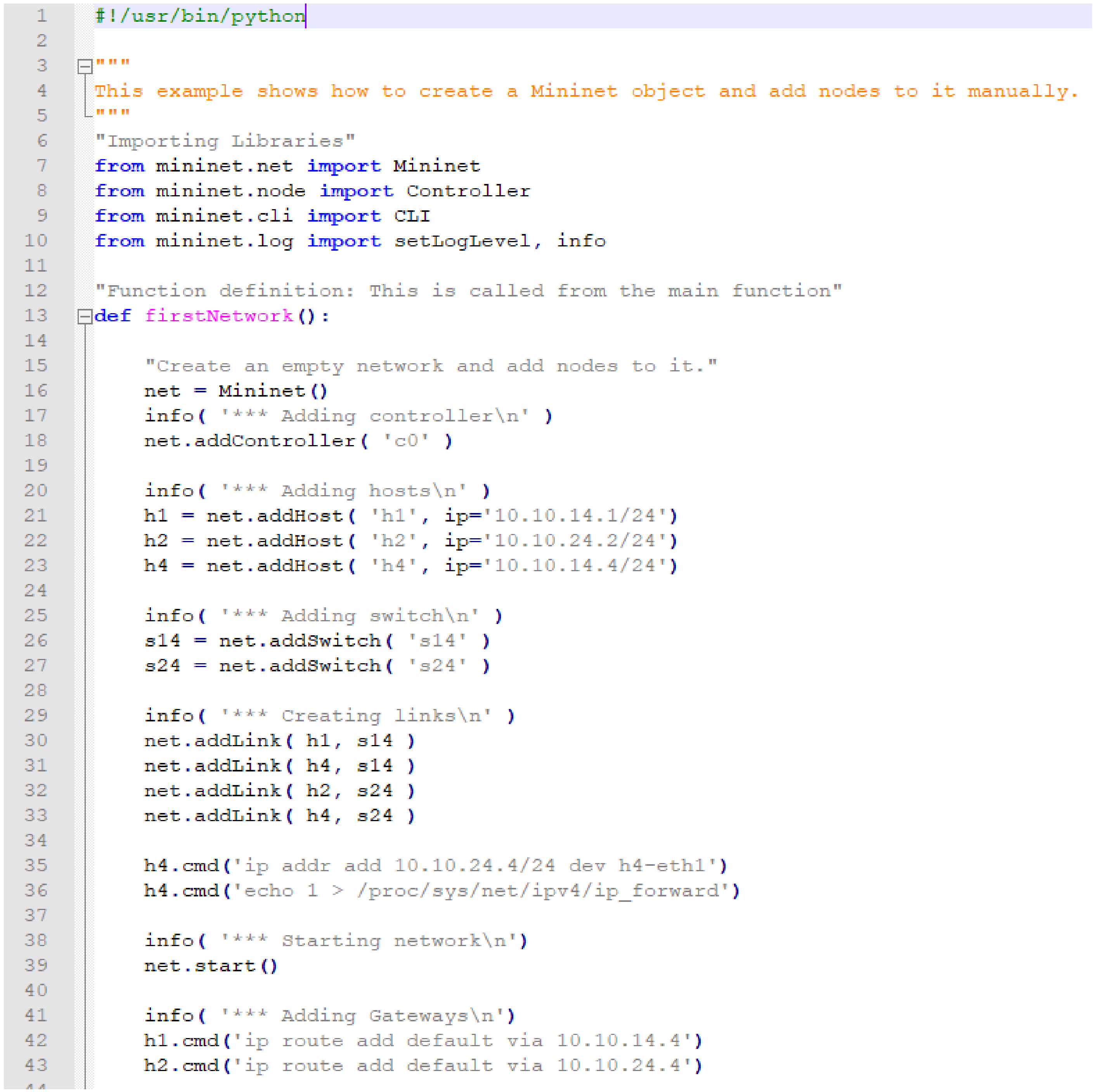
Man-in-the-middle attacks (MITM) are a common type of cybersecurity attack that allows attackers to eavesdrop on the communication between two targets. The attack takes place in between two legitimately communicating hosts, allowing the attacker to “listen” to a conversation they should normally not be able to listen to, hence the name “man-in-the-middle.”


User 1
Direct network traffic interrupted
User 2
Network traffic diverted through attacker
User 1’s computer thinks it’s talking to User 2’s computer
User 2’s computer thinks it’s talking to User 1’s computer
Man in the Middle Attacker

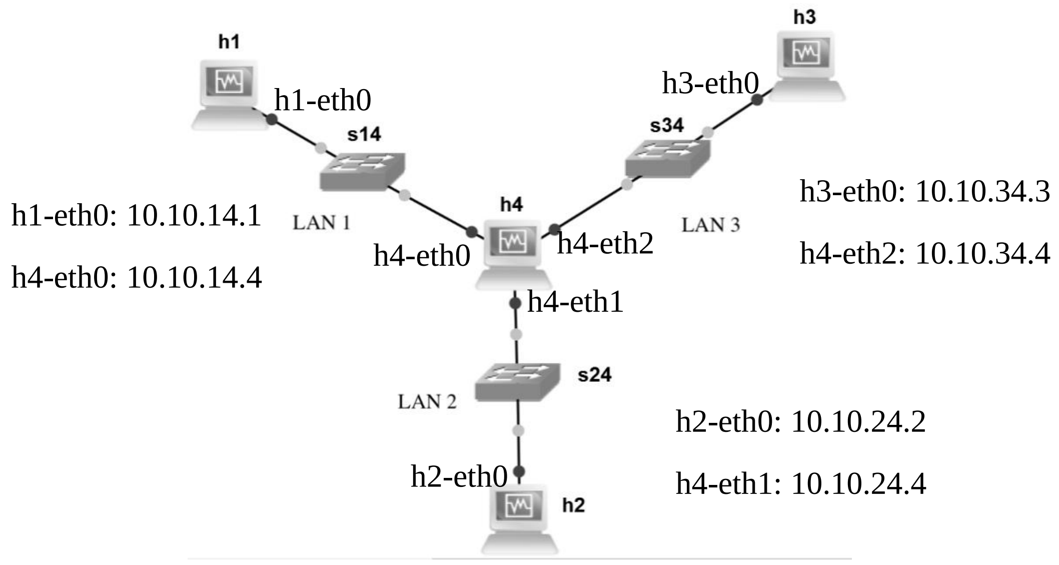
IP spoofing

- IP spoofing is a method adopted by attackers to send forged address in their attack traffic:
    - i.e., they can send an IP packet with an IP address of their wish!
- Most of the times, spoofing is used by an attacker mainly for the following reasons:
    - To conduct a DDoS (Distributed Denial of Service) attack, and he does not want the response from the target machine to reach him.
    - To compromise source-based authentication.


h1
Bank
s14
LAN 1
h4
Home Router
LAN 2
s24
Bob
h2
h3
Attacker (You)
s34
LAN 3



```python
#!/usr/bin/python

"""
This example shows how to create a Mininet object and add nodes to it manually.
"""
"Importing Libraries"
from mininet.net import Mininet
from mininet.node import Controller
from mininet.cli import CLI
from mininet.log import setLogLevel, info

"Function definition: This is called from the main function"
def firstNetwork():

    "Create an empty network and add nodes to it."
    net = Mininet()
    info( '*** Adding controller\n' )
    net.addController( 'c0' )

    info( '*** Adding hosts\n' )
    h1 = net.addHost( 'h1', ip='10.10.14.1/24')
    h2 = net.addHost( 'h2', ip='10.10.24.2/24')
    h4 = net.addHost( 'h4', ip='10.10.14.4/24')

    info( '*** Adding switch\n' )
    s14 = net.addSwitch( 's14' )
    s24 = net.addSwitch( 's24' )

    info( '*** Creating links\n' )
    net.addLink( h1, s14 )
    net.addLink( h4, s14 )
    net.addLink( h2, s24 )
    net.addLink( h4, s24 )

    h4.cmd('ip addr add 10.10.24.4/24 dev h4-eth1')
    h4.cmd('echo 1 > /proc/sys/net/ipv4/ip_forward')

    info( '*** Starting network\n')
    net.start()

    info( '*** Adding Gateways\n')
    h1.cmd('ip route add default via 10.10.14.4')
    h2.cmd('ip route add default via 10.10.24.4')
```



h1-eth0: 10.10.14.1

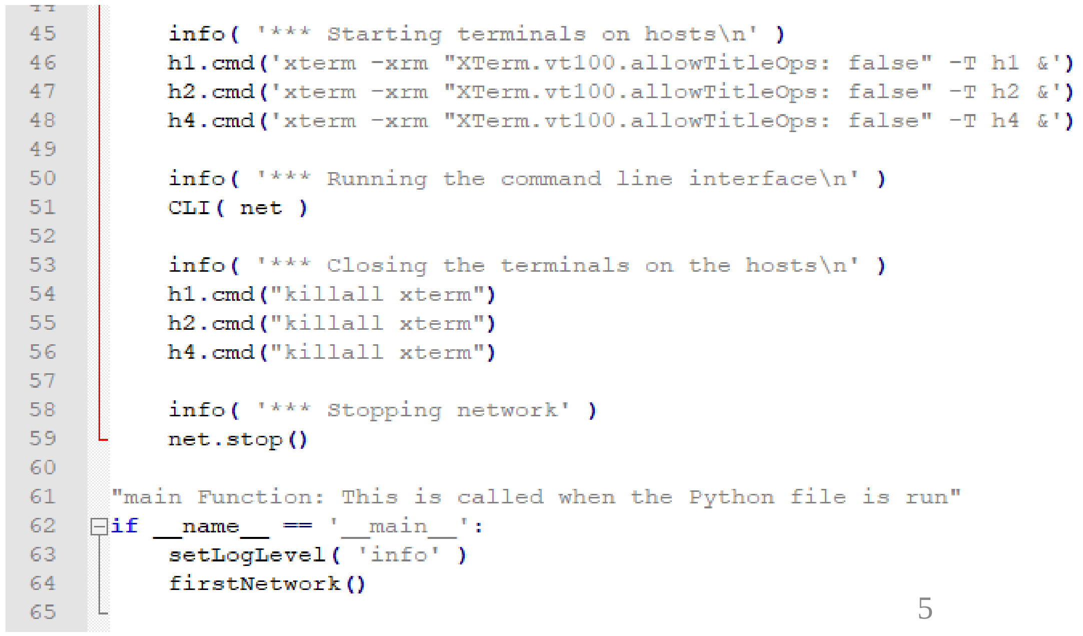
h4-eth0: 10.10.14.4

h3-eth0: 10.10.34.3

h4-eth2: 10.10.34.4

h2-eth0: 10.10.24.2

h4-eth1: 10.10.24.4



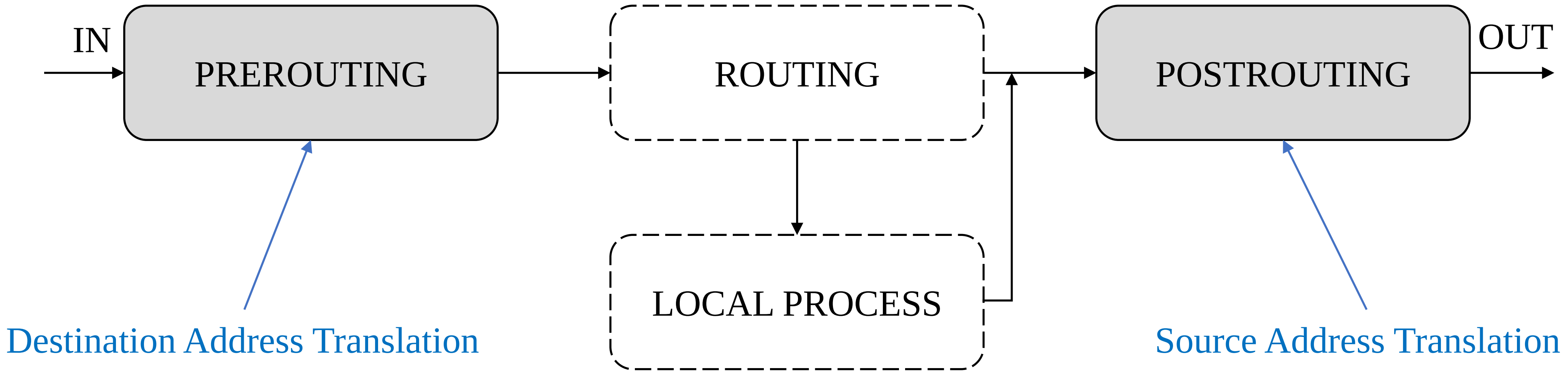
```python
    info( '*** Starting terminals on hosts\n' )
    h1.cmd('xterm -xrm "XTerm.vt100.allowTitleOps: false" -T h1 &')
    h2.cmd('xterm -xrm "XTerm.vt100.allowTitleOps: false" -T h2 &')
    h4.cmd('xterm -xrm "XTerm.vt100.allowTitleOps: false" -T h4 &')

    info( '*** Running the command line interface\n' )
    CLI( net )

    info( '*** Closing the terminals on the hosts\n' )
    h1.cmd("killall xterm")
    h2.cmd("killall xterm")
    h4.cmd("killall xterm")

    info( '*** Stopping network' )
    net.stop()

"main Function: This is called when the Python file is run"
if __name__ == '__main__':
    setLogLevel( 'info' )
    firstNetwork()
```

iptables

- The Linux kernel contains a packet filter framework called **netfilter** which enables a Linux machine to use rule chains and configure the IP packets. When a connection tries to establish itself on your system, iptables looks for a rule in its list to match it to. If it doesn't find one, it resorts to the default action.
- The three types of iptables:
  1. Mangle: to manage class-based queuing, modify QoS, TTL, …
  2. NAT: to change the IP addresses of the packets
  3. Filter: to accept or drop packets
- iptables command:
  - $ iptables -t [table] […]

NAT table

- This table has two types of rule chains:
  1. PREROUTING: to modify packets as soon as they arrive at the computer
  2. POSTROUTING: to modify packets that are ready to leave the computer



Destination NAT

- PREROUTING rules are used for Destination NAT
- # iptables -t nat -A (-I) PREROUTING [match pattern] -j [action]
  - -A: Append a rule at the end of the PREROUTING chain
  - -I: Insert a rule at the beginning of the PREROUTING chain
  - [match pattern]:
    - -p [protocol]: -p icmp, -p tcp, -p udp, …
    - -s [source_ip]: -s 192.168.1.1
    - -d [destination_ip]: -d 192.168.2.2
    - -i [incoming_interface_name]: -i h1-eth0, -i h4-eth2
    - (Only for tcp & udp:) --dport [destination_port_number]: --dport 80
  - [action]:
    - DNAT --to [desired_destination_ip]

DNAT examples

- *Change destination of TCP packets from 1.1.1.1 into 3.3.3.3:*

```
# iptables -t nat -A PREROUTING -p tcp -s 1.1.1.1 -j DNAT --to 3.3.3.3
```

- *Change destination of TCP packets to 2.2.2.2 into 3.3.3.3:*

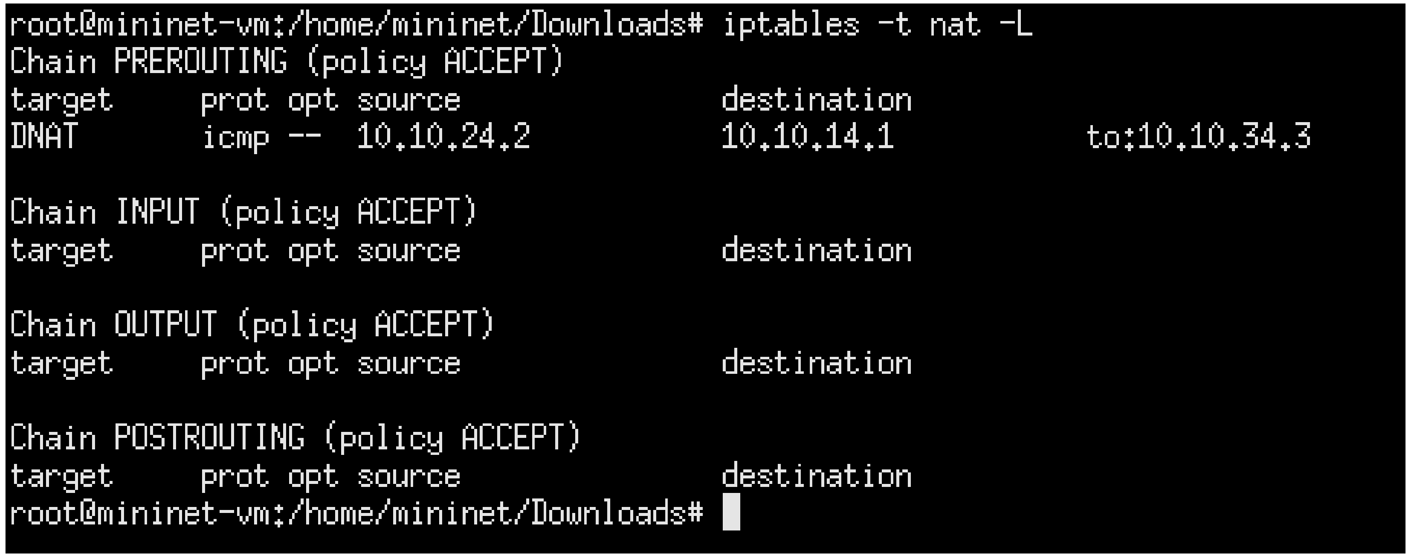
```
# iptables -t nat -A PREROUTING -p tcp -d 2.2.2.2 -j DNAT --to 3.3.3.3
```

- *Change destination of packets from 1.1.x.x to 2.2.2.2 into 3.3.3.3:*

```
# iptables -t nat -A PREROUTING -s 1.1.0.0/16 -d 2.2.2.2 -j DNAT --to 3.3.3.3
```

Rule chains

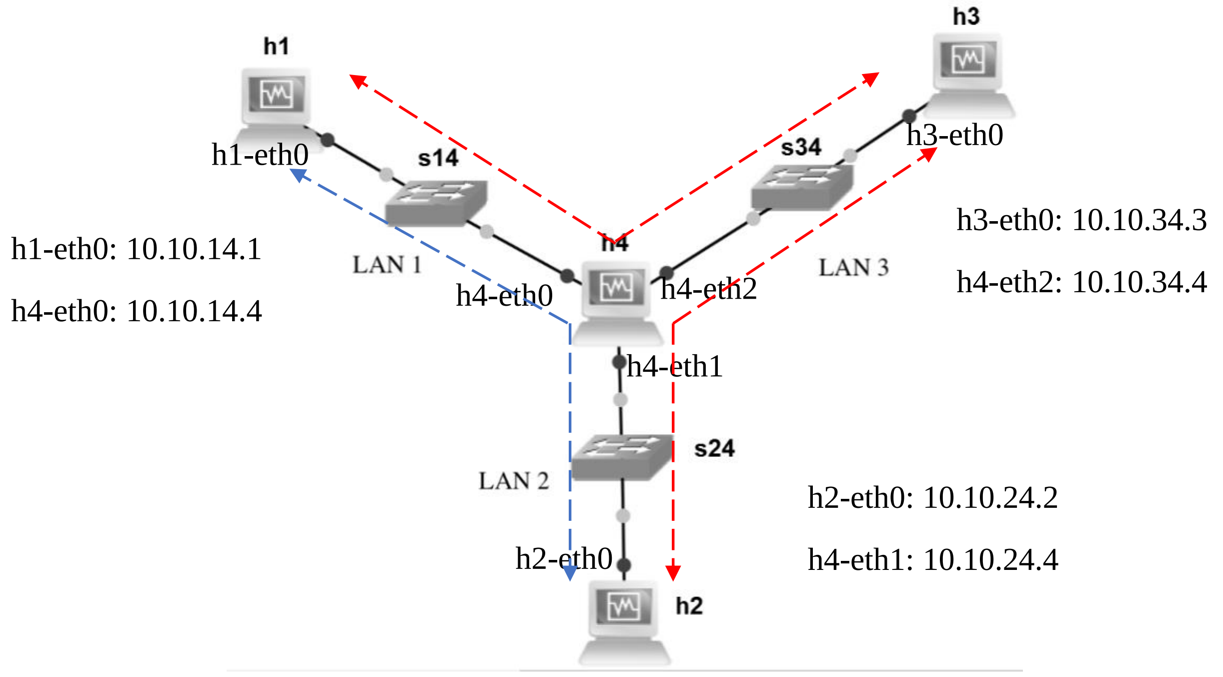
- List rules:
  - # iptables -t nat -L
- Flush (remove) all rules in a chain:
  - # iptables -t nat -F PREROUTING (POSTROUTING)
- Flush (remove) all rules:
  - # iptables -t nat -F
  - # iptables -F



```
root@mininet-vm:/home/mininet/Downloads# iptables -t nat -L
Chain PREROUTING (policy ACCEPT)
target     prot opt source               destination
DNAT       icmp --  10.10.24.2           10.10.14.1           to:10.10.34.3

Chain INPUT (policy ACCEPT)
target     prot opt source               destination

Chain OUTPUT (policy ACCEPT)
target     prot opt source               destination

Chain POSTROUTING (policy ACCEPT)
target     prot opt source               destination
root@mininet-vm:/home/mininet/Downloads#
```

Man-in-the-middle Attack

- Let's assume that a successful ping between two devices is equivalent to a successful money transfer between them.
- # iptables -t nat -A PREROUTING -p [protocol] -s [source_ip] -d [destination_ip] -j DNAT --to [desired_destination_ip]

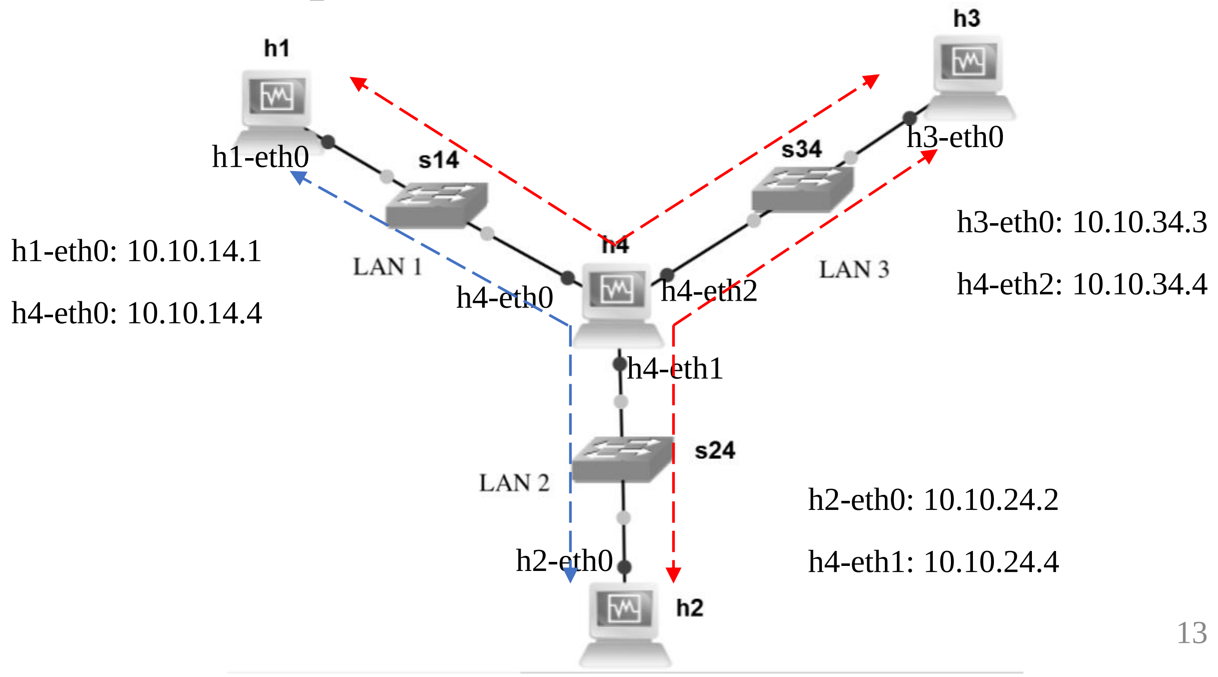


Reverse Path Filtering (RPF)

- Reverse path filtering is a mechanism adopted by the Linux kernel, as well as most of the networking devices out there to check whether a receiving packet source address is routable.
- So in other words, when a machine with reverse path filtering enabled receives a packet, the machine will first check whether the source of the received packet is reachable through the interface it came in.
  - If it is routable through the interface which it came, then the machine will accept the packet.
  - If it is not routable through the interface which it came, then the machine will drop that packet.

Reverse Path Filtering (RPF)

- Basically, if the reply to this packet wouldn't go out the interface this packet came in, then this is a bogus packet and should be ignored.
- Disable RPF for an interface, e.g. h4-eth0:
  - # echo 0 > /proc/sys/net/ipv4/conf/h4-eth0/rp_filter
- Disable RPF for h4:
  - # sudo sh disableRPF.sh



Source NAT

- POSTROUTING rules are used for Source NAT
- # iptables -t nat -A (-I) POSTROUTING [match pattern] -j [action]
  - [match pattern]:
    - -p [protocol]: -p icmp, -p tcp, -p udp, …
    - -s [source_ip]: -s 192.168.1.1
    - -d [destination_ip]: -d 192.168.2.2
    - -o [outgoing_interface_name]: -o h1-eth0, -o h4-eth2
    - (Only for tcp & udp:) --sport [source_port_number]: --sport 80
  - [action]:
    - SNAT --to [desired_source_ip]
    - MASQUERADE
      - Source IP is replaced by the source of the host machine
      - MASQUERADE ≡ SNAT --to [host_ip]

SNAT examples

- *Change source of packets from 1.1.1.1 leaving at h4-eth0 into 3.3.3.3:*

```
# iptables -t nat -A POSTROUTING -o h4-eth0 -s 1.1.1.1 -j SNAT --to 3.3.3.3
```

- *Change source of packets to 2.2.2.2 leaving at h4-eth0 into 3.3.3.3:*

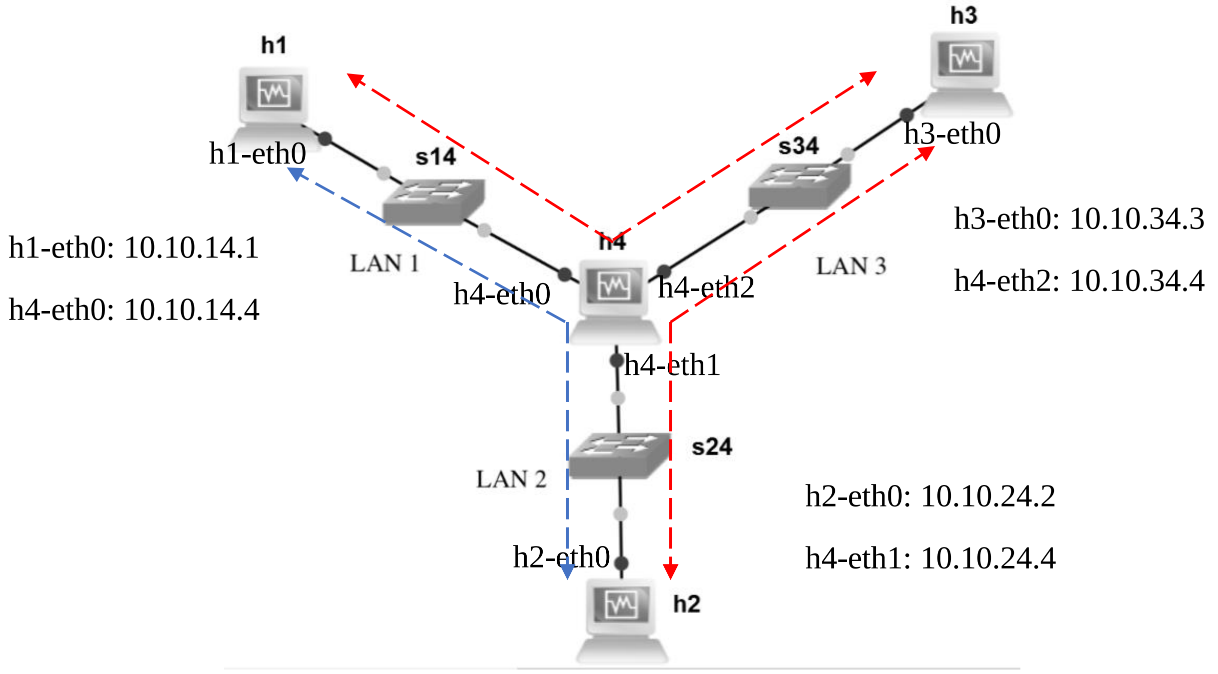
```
# iptables -t nat -A POSTROUTING -o h4-eth0 -d 2.2.2.2 -j SNAT --to 3.3.3.3
```

- *Change source of packets from 1.1.x.x to 2.2.2.2 into 3.3.3.3:*

```
# iptables -t nat -A POSTROUTING -s 1.1.0.0/16 -d 2.2.2.2 -j SNAT --to 3.3.3.3
```

Man-in-the-middle Attack

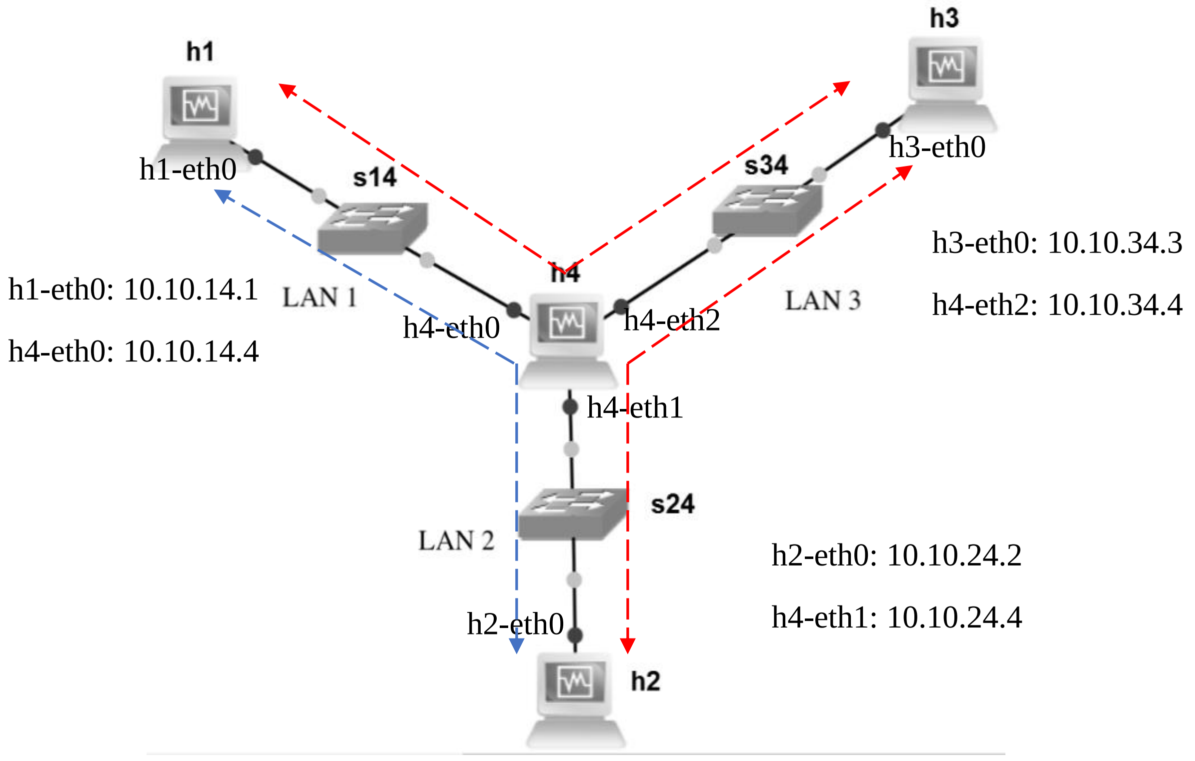
- If RPF is enabled:
- # iptables -t nat -A POSTROUTING -o [outgoing_interface_name] -s [source_ip] -d [destination_ip] -j SNAT --to [desired_source_ip]



Questions

- What about tampering the IP routing table?

- How can h2 notice the attack?


h1
h3
h1-eth0
s14
s34
h3-eth0
h3-eth0: 10.10.34.3
h4-eth2: 10.10.34.4
h1-eth0: 10.10.14.1
h4-eth0: 10.10.14.4
LAN 1
h4
LAN 3
h4-eth0
h4-eth2
h4-eth1
s24
LAN 2
h2-eth0: 10.10.24.2
h4-eth1: 10.10.24.4
h2-eth0
h2