

# LAN PROTOCOL ATTACKS

## PART 1 - ARP RELOADED

Presented at Defcon 15  
Las Vegas, NV 2007

Jesse 'x30n' D'Aguzzo  
[jesse\[at\]praetoriang.net](mailto:jesse@praetoriang.net) / [x30n\[at\]digrev.org](mailto:x30n[at]digrev.org)

# Background



PRAETORIAN  
GLOBAL™

*Proven Information Security*

- \$whoami
  - Jesse 'x30n' D'Aguanno
  - Director of Professional Services and Research - Praetorian Global
    - Lead Professional Services - Penetration Testing, Code audit, etc.
    - Lead Vulnerability Research Efforts
  - Team Captain / Researcher - Digital Revelation
    - Group of hackers who share stuff & beat other hackers in competitions
    - Usual suspects in Defcon CTF
    - Taken Defcon CTF title twice + Interz0ne CTF & toorcon rootwars
  - Invented techniques and coined the term “Blackjacking”
  - Didn't write the book “Blackjacking”

# ARP - Background



- Networked systems need a way to identify the layer 2 address of peers on the same network segment based on their protocol address
- Enter the Address Resolution Protocol (ARP)
  - ARP allows a system to dynamically identify the hardware address (In our case MAC address) of a networked peer via its protocol address (IP address)
  - It does this by sending REQUEST packets to the network broadcast for any one with the IP we're looking for
    - I.e "Who has 192.168.1.100"
  - The host with the matching IP sends a RESPONSE packet to the requester with it's hardware address
    - I.e "192.168.1.100 is at 00:DF:1A:9C:A3:78"
  - To reduce the amount of transactions required, hosts maintain an "ARP cache" of recently resolved MAC Address / IP Pairs
    - Else, each transaction would require an ARP lookup

# ARP - Background



- Anatomy of an ARP packet

BITS 0 - 7	8 - 15	16 - 31
<b>HTYPE</b> HARDWARE TYPE		<b>PTYPE</b> PROTOCOL TYPE
<b>HLEN</b> HW LENGTH	<b>PLEN</b> PROTO LENGTH	<b>OPER</b> OPERATION CODE
<b>SHA</b> SENDER HARDWARE ADDRESS		
<b>SPA</b> SENDER PROTOCOL ADDRESS		
<b>THA</b> TARGET HARDWARE ADDRESS		
<b>TPA</b> TARGET PROTOCOL ADDRESS		



# ARP Request

- 192.168.1.20 Requests the hardware address of 192.168.1.100

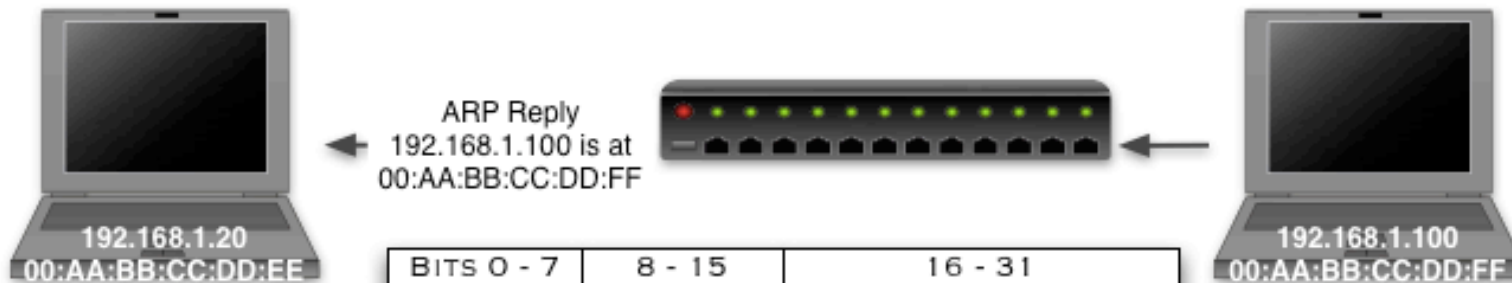


ARP Request  
Who has 192.168.1.100?  
Tell 192.168.1.20



BITS 0 - 7	8 - 15	16 - 31
<b>0x01</b> ETHERNET		<b>0x0800</b> IP
<b>0x06</b> ETHERNET LENGTH	<b>0x04</b> IPv4 LENGTH	<b>0x01</b> REQUEST
<b>00:AA:BB:CC:DD:EE</b> SENDER HARDWARE ADDRESS		
<b>0xCOA80114</b> 192.168.1.20		
<b>00:00:00:00:00:00</b> TARGET HARDWARE ADDRESS		
<b>0xCOA80164</b> 192.168.1.100		

# ARP - Reply



BITS 0 - 7	8 - 15	16 - 31
<b>0x01</b> ETHERNET		<b>0x0800</b> IP
<b>0x06</b> ETHERNET LENGTH	<b>0x04</b> IPV4 LENGTH	<b>0x02</b> REPLY
<b>00:AA:BB:CC:DD:FF</b> SENDER HARDWARE ADDRESS		
<b>0xC0A80164</b> 192.168.1.100		
<b>00:AA:BB:CC:DD:EE</b> TARGET HARDWARE ADDRESS		
<b>0xC0A80114</b> 192.168.1.20		

# ARP Attacks - ARP Cache



PRAETORIAN  
GLOBAL™

*Proven Information Security*

- ARP Cache Poisoning
  - ARP is unauthenticated
  - Whoever sends a reply to a requester first wins
    - The ARP cache is updated with the contents of the reply
    - Sometimes the ARP cache is updated when it gets a reply even if it didn't send a request
      - Depends on OS
  - Traditionally, this is exploited by sending forged REPLY packets to a victim to have them update their ARP cache

# ARP Attacks - ARP Cache



PRAETORIAN  
GLOBAL™

*Proven Information Security*

- ARP cache poisoning results:
  - Man in the Middle
    - E.g Poison the victim with your MAC address for the IP address of the gateway... Host communicates with you instead of the gateway
  - Denial of Service (DoS)
    - E.g Poison the victim with a bogus MAC address and the IP address of the gateway... Host can't communicate with gateway



# ARP Cache Poisoning - MiTM

- Client Sends ARP Request for 192.168.1.1 (Gateway)



ARP Request  
Who has 192.168.1.1?  
Tell 192.168.1.20

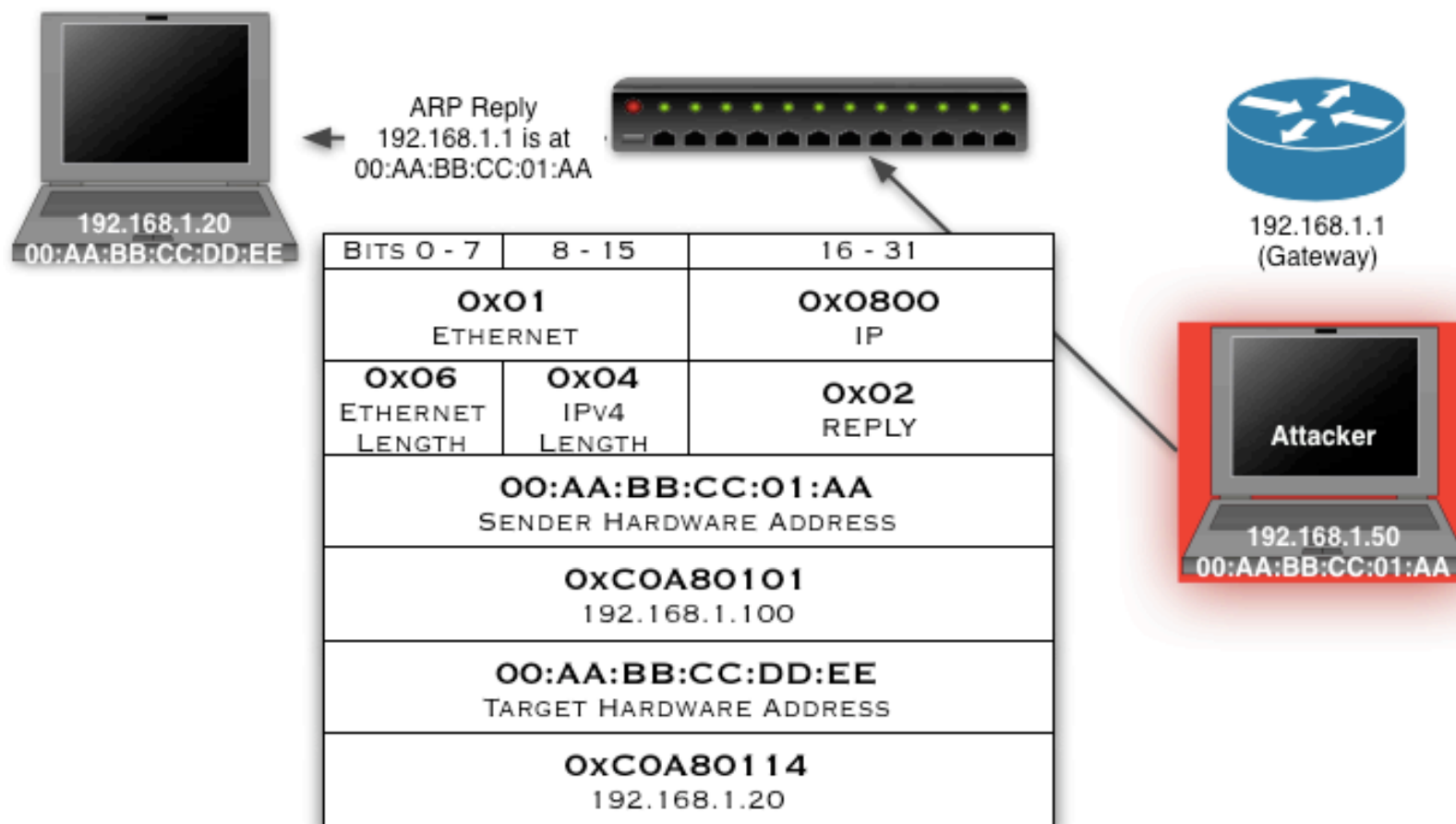


BITS 0 - 7	8 - 15	16 - 31
0x01 ETHERNET		0x0800 IP
0x06 ETHERNET LENGTH	0x04 IPV4 LENGTH	0x01 REQUEST
00:AA:BB:CC:DD:EE SENDER HARDWARE ADDRESS		
0xCOA80114 192.168.1.20		
00:00:00:00:00:00 TARGET HARDWARE ADDRESS		
0xCOA80101 192.168.1.1		



# ARP Cache Poisoning - MiTM

- Attacker sends spoofed REPLY packet to host with its MAC for 192.168.1.1





# ARP Cache Poisoning - MiTM

- Attacker can now forward packets to 192.168.1.1 intercepting all communication between 192.168.1.20 and the Gateway (192.168.1.1)





# ARP Cache Poisoning - DoS

- 192.168.1.20 Requests the hardware address of 192.168.1.1 (Gateway)



ARP Request  
Who has 192.168.1.1?  
Tell 192.168.1.20

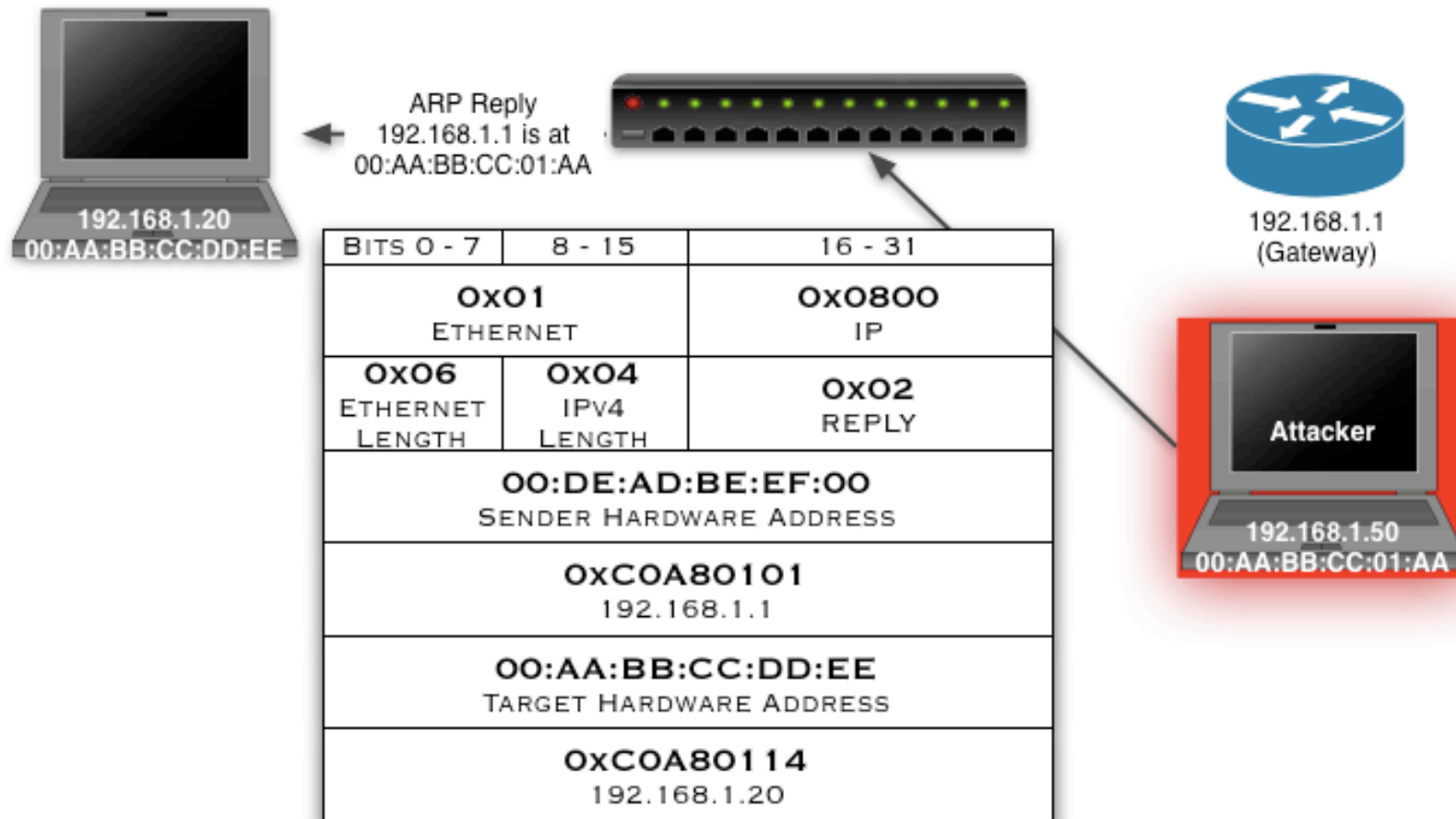


BITS 0 - 7	8 - 15	16 - 31
<b>0x01</b> ETHERNET		<b>0x0800</b> IP
<b>0x06</b> ETHERNET LENGTH	<b>0x04</b> IPV4 LENGTH	<b>0x01</b> REQUEST
<b>00:AA:BB:CC:DD:EE</b> SENDER HARDWARE ADDRESS		
<b>0xCOA80114</b> 192.168.1.20		
<b>00:00:00:00:00:00</b> TARGET HARDWARE ADDRESS		
<b>0xCOA80101</b> 192.168.1.1		



# ARP Cache Poisoning - DoS

- Attacker sends spoofed REPLY packet to 192.168.1.1 that 192.168.1.1 is at 00:DE:AD:BE:EF:01 (Non-existent MAC Address)



# ARP Cache Poisoning - DoS



- Victim now thinks 192.168.1.1 is at 00:DE:AD:BE:EF:01 so communication is broken



# ARP Attacks - CAM Table Overflow



- CAM Table Overflow
  - A network hub broadcasts all packets to *all* ports
  - A network switch on the other hand maintains a record of what hardware addresses are associated with each switch port
    - Stores this and other info in its CAM (Content-Addressable Memory) table
  - Only forwards packets to the port associated with the destination hardware address in the ethernet frame
  - CAM table overflow
    - Attacker sends thousands of bogus MAC addresses to the network
    - Switch's CAM table is updated with each MAC
    - The CAM table can only hold so much data, so at some point it becomes full
    - The switch then forwards all traffic to all ports again (Like a hub)
- CAM table overflow results:
  - Since switch now acts the same as a hub, the attacker can eavesdrop (“sniff”) all traffic on that network segment

# ARP Attacks - Weaknesses



- ARP cache poisoning
  - Current methods of ARP cache poisoning utilize ARP REPLY packets
  - Unfortunately, not all OSs will update their ARP cache when they receive an ARP REPLY if they haven't first sent a matching ARP REQUEST
  - To overcome this, some poisoning tools sniff the network for broadcast ARP REQUESTs, sending forged REPLYs in response to all requests
- CAM table overflow
  - CAM table overflow requires many hundreds if not thousands of spoofed ARPs
  - Obviously very noisy and identifiable attack
  - Technologies like Cisco's port security can limit the number of MAC addresses allowed on each switch port
    - Mitigates the risk of a CAM table overflow since the switch doesn't recognize any more MAC addresses on that port once the pre-set limit is reached

# Arpcraft



- arpcraft tool was created to help in testing different ARP conditions

```
~$./arpcraft
ERROR: Missing required options!
Usage: ./arpcraft [options]
Options:
-i <interface>           Interface to send on      *
-sha <MAC Address>      Source Hardware Address *
-spa <IP Address>       Source Protocol Address *
-tha <MAC Address>      Target Hardware Address *
-tpa <IP Address>       Target Protocol Address *
-o "request" or "reply" Opcode                  *
-esrc <MAC Address>     Source MAC for Ethernet Frame
-edst <MAC Address>     Destination MAC for Ethernet Frame
-h                       This Help Message
-interval <seconds>
                        How often we should send ARP (defaults to 5 secs)
-c -count <rounds>
                        How many packets to send (Defaults to unlimited)
```

# Arpcraft



- arpcraft (cont.)

```

                                Ethernet Frame Layout
#####
# Destination MAC Address:      00:00:00:00:00:00      #
# Source MAC Address:         00:00:00:00:00:00      #
# Ether Type:                 ARP                    #
#####
                                ARP Packet Layout
#####
# Hardware Type:              1 (Ethernet)           #
# Protocol Type:              0800 (IP)              #
# Hardware Address Length:    (6)                   #
# Protocol Address Length:    (4)                   #
# Opcode:                    <Unknown Opcode>       #
# Sender Hardware Address (SHA): 00:00:00:00:00:00   #
# Sender Protocol Address (SPA): 127.0.0.1          #
# Target Hardware Address (THA): 00:00:00:00:00:00   #
# Target Protocol Address (TPA): 127.0.0.1          #
#####
~$
```

# Arpcraft



- Examples

- Send ARP Request to 192.168.1.100 [00:AA:BB:CC:DD:FF] from 192.168.1.20 [00:AA:BB:CC:DD:EE]

```
~$./arpcraft -i en0 -sha 00:aa:bb:cc:dd:ee -spa 192.168.1.20 -tha  
00:aa:bb:cc:dd:ff -tpa 192.168.1.100 -o request
```

### Ethernet Frame Layout

```
#####  
# Destination MAC Address:      00:aa:bb:cc:dd:ff      #  
# Source MAC Address:          00:aa:bb:cc:dd:ee      #  
# Ether Type:                   ARP                    #  
#####
```

### ARP Packet Layout

```
#####  
# Hardware Type:                 1 (Ethernet)          #  
# Protocol Type:                 0800 (IP)             #  
# Hardware Address Length:       (6)                  #  
# Protocol Address Length:       (4)                  #  
# Opcode:                         Request             #  
# Sender Hardware Address (SHA):  00:aa:bb:cc:dd:ee     #  
# Sender Protocol Address (SPA):  192.168.1.20         #  
# Target Hardware Address (THA):  00:aa:bb:cc:dd:ff     #  
# Target Protocol Address (TPA):  192.168.1.100        #  
#####
```

```
Injecting ARP request to 192.168.1.100 [00:aa:bb:cc:dd:ff] from 192.168.1.20  
[00:aa:bb:cc:dd:ee]
```

# Arpcraft



- Send ARP Reply to 192.168.1.85 [00:AB:CD:EF:01:02] from 192.168.1.20 [00:AA:BB:CC:DD:EE]

```
~$./arpcraft -i en0 -sha 00:aa:bb:cc:dd:ee -spa 192.168.1.20 -tha 00:AB:CD:EF:01:02  
-tpa 192.168.1.85 -o reply
```

## Ethernet Frame Layout

```
#####  
# Destination MAC Address:      00:AB:CD:EF:01:02      #  
# Source MAC Address:          00:aa:bb:cc:dd:ee      #  
# Ether Type:                  ARP                    #  
#####
```

## ARP Packet Layout

```
#####  
# Hardware Type:                1 (Ethernet)          #  
# Protocol Type:                0800 (IP)             #  
# Hardware Address Length:      (6)                   #  
# Protocol Address Length:      (4)                   #  
# Opcode:                       Reply                 #  
# Sender Hardware Address (SHA): 00:aa:bb:cc:dd:ee    #  
# Sender Protocol Address (SPA): 192.168.1.20        #  
# Target Hardware Address (THA): 00:AB:CD:EF:01:02    #  
# Target Protocol Address (TPA): 192.168.1.85        #  
#####
```

```
Injecting ARP reply to 192.168.1.85 [00:AB:CD:EF:01:02] from 192.168.1.20  
[00:aa:bb:cc:dd:ee]
```



# Attacks Reloaded - ARP Cache

- Again, traditional ARP cache poisoning techniques have weaknesses
  - Some OSs don't update their ARP cache after receiving gratuitous ARP Replies that they aren't expecting
  - Even if they do, most (All?) OSs will not add an entry to their ARP cache after receiving a Reply if an entry isn't already there
- RFC 826 [1] states:
  - If a host receives an ARP *REQUEST*
  - and the target hardware address (THA) and target protocol address (TPA) match their own (ARP request destined for them)
  - Prior to responding, the host will update or *add* the source protocol address (SPA) and source hardware address (SHA) to its cache
  - Actual verbiage from RFC 826:
    - “... If the pair <protocol type, sender protocol address> is already in my translation table, update the sender hardware address field of the entry with the new information in the packet and set Merge\_flag to true.
    - Am I the target protocol address?
      - Yes:
        - If Merge\_flag is false, add the triplet <protocol type, sender protocol address, sender hardware address> to the translation table...”

# Attacks Reloaded - ARP Cache



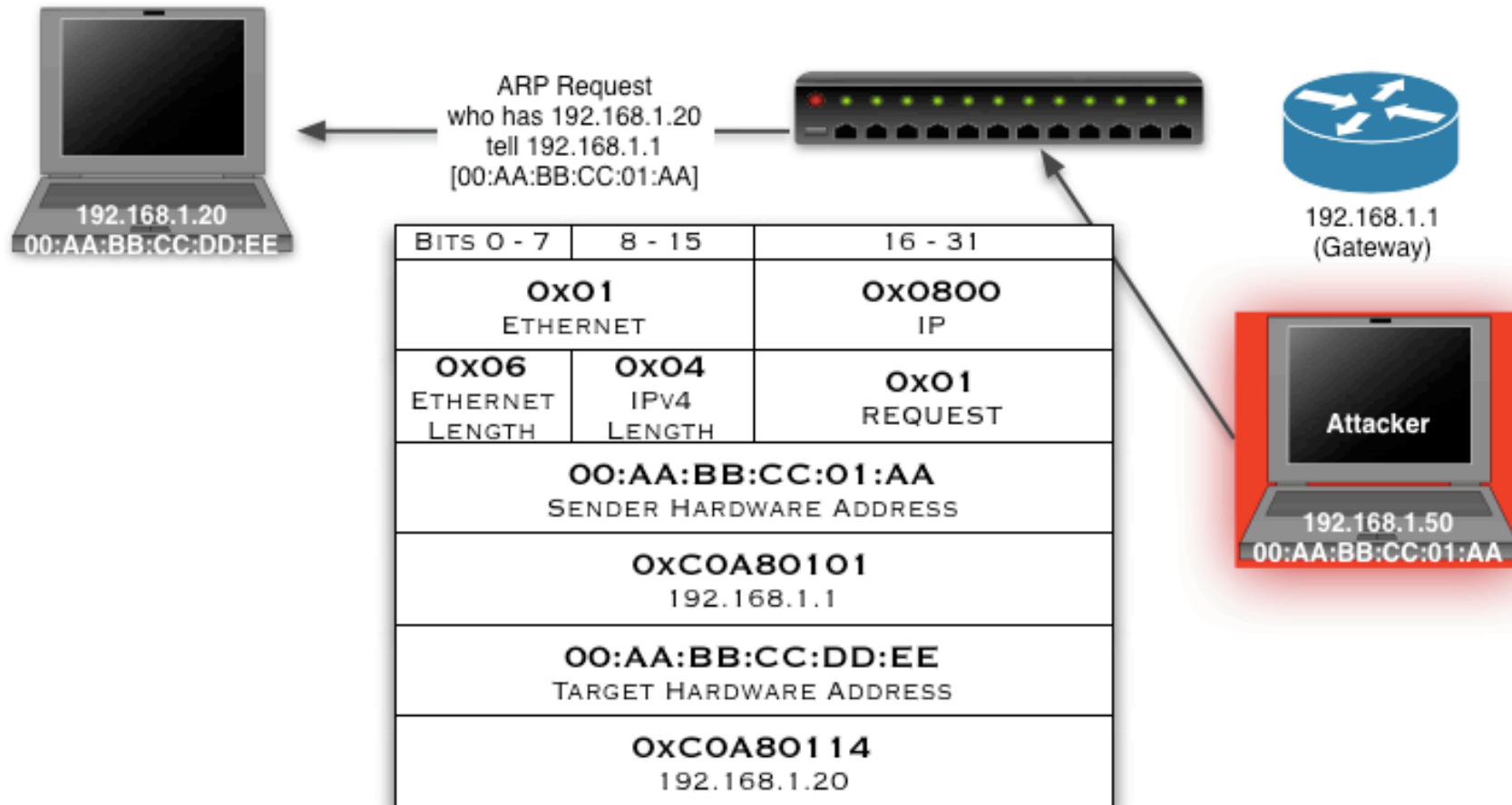
**PRAETORIAN  
GLOBAL™**  
*Proven Information Security*

- So according to RFC 826, if we use an ARP REQUEST packet instead of an ARP REPLY, we can poison our victim's cache every time with a single packet
  - Even adding new entries to the cache instead of relying on there already being a matching entry



# Attacks Reloaded - ARP Cache

- Attacker sends REQUEST to 192.168.1.20 from 192.168.1.1 with attacker's MAC



# Attacks Reloaded - ARP Cache



**PRAETORIAN  
GLOBAL™**  
*Proven Information Security*

- Demo

# Attacks Reloaded - Focused



- As mentioned, CAM table overflow technique has drawbacks
  - Highly identifiable
  - Hundreds or Thousands of packets
  - Modern switches can defeat it with technology like “Port Security”
    - Limit the number of MAC addresses allowed to be associated with a given port
- Often, we want to sniff the traffic of a particular host, not necessarily the whole network
- When the switch receives a frame it checks its CAM table to see if the destination hardware address exists
- If not, the switch forwards the frame to all ports
- We can accomplish this in a different way than CAM table overflow

# Attacks Reloaded - Focused



- Utilizing the ARP cache poisoning technique detailed above, if we poison the ARP cache of the victim with a NULL hardware address (00:00:00:00:00:00) for the destination (I.e the network gateway)
- All frames from the victim to the gateway will be forwarded to all switch ports, allowing us to eavesdrop on that communication
  - Works on every switch we've tested:
    - Cisco
    - Bay Networks
    - HP
    - Etc.
- Works despite Port Security, etc. because we're sending 1 packet (Or at least one MAC, 00:00:00:00:00:00) as opposed to thousands
- Also, much less identifiable to network detection tools (IDS, etc)

# Attacks Reloaded - Focused



**PRAETORIAN  
GLOBAL™**  
*Proven Information Security*

- Demo

# Attacks Reloaded - Other Fun



- Some OSs can be poisoned with their own MAC address for the destination

- Severs communication
- Prevents messages from ever reaching the network
- Useful for:
  - Preventing log messages from reaching log host
  - Preventing host based IDS from reporting to central console
  - etc.

– Victim = 192.168.1.20 [00:AA:BB:CC:DD:EE]

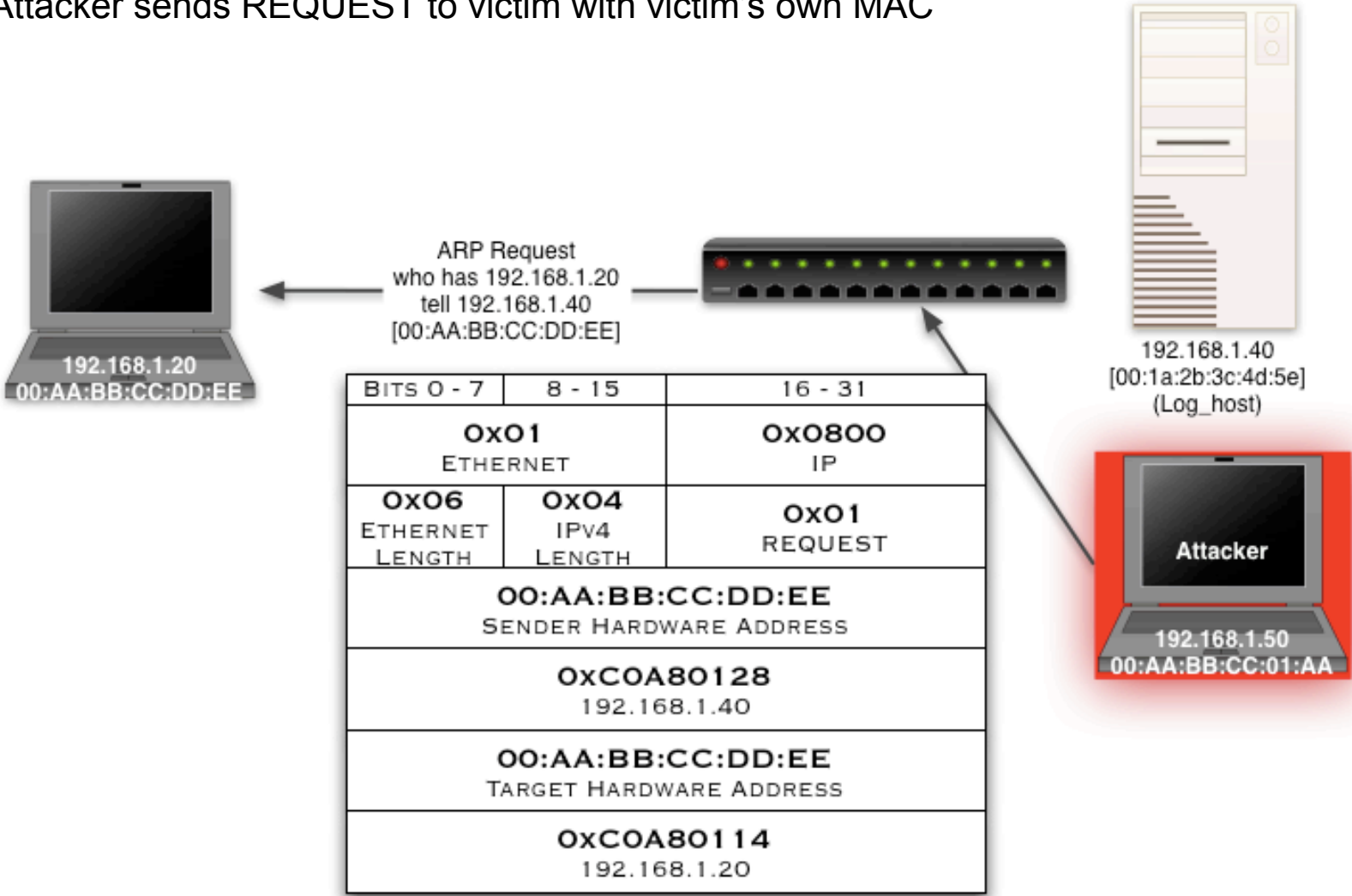
– Log\_host = 192.168.1.40 [00:1A:2B:3C:4D:5E]

```
attacker~$./arpcraft -i fxp0 -sha 00:aa:bb:cc:dd:ee -spa 192.168.1.40  
-tha 00:aa:bb:cc:dd:ee -tpa 192.68.1.20 -o request
```



# Attacks Reloaded - Other Fun

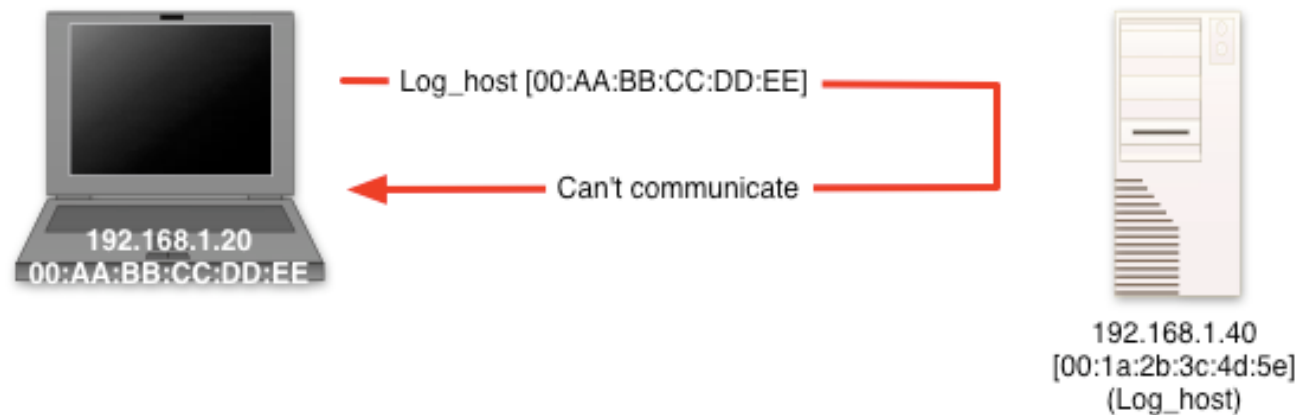
- Attacker sends REQUEST to victim with victim's own MAC



# Attacks Reloaded - Other Fun



- Victim attempts to log subsequent attack attempt
  - Victim's ARP cache says Log\_host is at local interface
  - Log message never reaches Log\_host



# Attacks Reloaded - Other Fun



- Duplicate IP address on network
  - If a host receives an ARP request or reply with the same protocol address (IP address) as its own interface, but a different hardware (MAC) address
    - Most OSs will warn the user that a duplicate IP address is in use on the network
    - Example: Windows pop-up
      - “Windows - System Error”
      - There is an IP address conflict with another system on the network
    - Proven effective in social engineering scenarios
      - Call victim: “Hello, this is Mr. Tech from IT... It appears that we have a network issue with your machine... Give me your username / password...”
    - Potential DoS
      - Many packets like this can slow Windows down considerably, etc.

# Arpvenom



**PRAETORIAN  
GLOBAL™**  
*Proven Information Security*

- Tool to automate ARP attacks presented today
- Details...

# Conclusion



PRAETORIAN  
GLOBAL™

*Proven Information Security*

- Traditional ARP attacks work well but can have some weaknesses in certain environments
- In some cases, using ARP REQUEST packets instead of ARP REPLY packets for cache poisoning can be more reliable
- We can perform “focused sniffing” between hosts by poisoning the victim’s cache with a NULL MAC address for the intended destination
  - Less noisy than CAM table overflow
  - More reliable on switches with controls to prevent CAM table overflow

# Resources



PRAETORIAN  
GLOBAL™

*Proven Information Security*

- Most recent version of slides, arpcraft, arpvenom, etc:
  - <http://www.praetorianglobal.net/Presentations/arpreload.html>
  - or
  - <http://www.digrev.org>
- RFC 826 ("An Ethernet Address Resolution Protocol" [Plummer-1982]): <http://www.faqs.org/rfcs/rfc826.html>
- ...

# Questions?



**PRAETORIAN  
GLOBAL™**  
*Proven Information Security*

- **Thanks!**
- Comments: [jesse@praetoriang.net](mailto:jesse@praetoriang.net)